RL-TR-96-87
Final Technical Report
August 1996

MODEL ABSTRACTION TECHNIQUES

Computer Sciences Corporation

Frederick K. Frantz and A. John Ellor

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

Although this report references limited documents (*), listed on page 158, no limited information has been extracted.

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-96-87 has been reviewed and is approved for publication.

APPROVED:

ALEX F. SISTI
Project Engineer

FOR THE COMMANDER:

JOSEPH CAMERA
Technical Director
Intelligence & Reconnaissance Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/IRAE, 32 Hangar Road, Rome, NY 13441-4114. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1996	Final Sep 94 - Mar 96

4. TITLE AND SUBTITLE

MODEL ABSTRACTION TECHNIQUES

5. FUNDING NUMBERS

C - F30602-94-C-0272
PE - 62702F
PR - 4594
TA - 15
WU - L8

6. AUTHOR(S)

Frederick K. Frantz and A. John Ellor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Sciences Corporation
One Mony Plaza
Syracuse NY 13202

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory (IRAE)
32 Hangar Rd
Rome NY 13441-4114

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-96-87

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Alex F. Sisti/IRAE/(315)330-4518

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

In this report the findings of research into model abstraction techniques is documented. Model abstraction is a way of simplifying an underlying conceptual model on which a simulation is based while maintaining the validity of the simulation results with respect to the question being addressed by the simulation. Manipulation of models using abstractions is an important part of the model development process.

14. SUBJECT TERMS

Model abstraction, Hierachy of models, Metamodeling, Model Taxonomy

15. NUMBER OF PAGES

282

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASIFIED	UNCLASSIFIED	SAR

Table of Contents

Appendices

LIST OF FIGURES

LIST OF TABLES

1. Introduction and Summary

This Final Scientific and Technical Report, entitled "Model Abstraction Techniques," documents the results of a study conducted by Computer Sciences Corporation (CSC). This research effort was conducted under contract Number F30602-94-C-0272 to Rome Laboratory, funded under a Broad Agency Announcement entitled "Modeling and Simulation Techniques." As such, this report fulfills the requirements of Contract Data Requirements List (CDRL) Item A005.

In this report, we document the findings of our research into model abstraction techniques. Model abstraction is a way of simplifying an underlying conceptual model on which a simulation is based while maintaining the validity of the simulation results with respect to the question being addressed by the simulation. Manipulation of models using abstractions is an important part of the model development process. A more detailed discussion of the concept of model abstraction and the motivation for its study is provided in Section 2.

The next three sections include a discussion of various model abstraction techniques drawn from several disciplines. Section 3 provides a discussion of relevant background from the simulation domain. A brief review of previous discussions of abstraction techniques documented in simulation literature is provided in Section 3, along with a formal framework for discrete event simulation. Ongoing developments in the artificial intelligence subfield of qualitative reasoning have also provided a rich source of model abstraction techniques. These techniques have potential application in the discrete event simulation domain, and are described in Section 4. The techniques identified in Sections 3 and 4 are organized in a taxonomy of abstraction techniques. This taxonomy is defined in Section 5.

The taxonomy provides an organization within which to discuss the specific application of abstraction techniques to discrete event simulation models. The next four sections include discussion of specific applications of abstraction techniques. We analyzed three existing models to consider how abstraction techniques could be applied. In Section 6, we describe an analysis of the Reactive Wild Weasel Model (RWWM), an air/ground engagement model. We also analyzed a radar detection engineering model, ALARM, and document the results of the analysis in Section 7. Section 8 includes an analysis of a theater level model called the Advanced Regional Exploratory System (ARES). In Section 9, we illustrate additional abstraction applications applied to more generic C^3I modeling problems.

One observation that we made early in this study involves the synergism of abstraction techniques. We elaborate on several different ways in which abstraction techniques can be used cooperatively in Section 10. The main body of the report ends with our conclusions and recommendations, provided in Section 11. The report concludes with a list of references.

There are a number of appendices associated with this report. The following appendices provide supplemental and reference material that was generated as part of this research effort:

- Appendix A: Synopsis of the Discrete Event System Specification (DEVS) formalism, primarily drawn from (Zeigler 1984).
- Appendix B: Design documentation derived for analysis of the RWWM model.
- Appendix C: Detailed mathematical derivation of the RWWM launch point computation abstraction.
- Appendix D: Parameter dependency graphs for ALARM.
- Appendix E: Complete set of TERSM metamodel graphs, and an analysis of how the graphs were generated.
- Appendix F: Example of derivation of a qualitative model from a discrete event simulation model.
- Appendix G: Listings and notes for software written under this contract.

2. Setting the Stage

This section provides the context and foundation for the discussion of model abstraction techniques that are contained in the later sections of this report. We begin (Section 2.1) by introducing the concept of model abstraction and its role in the model development process. To assist the reader in understanding the application of model abstraction techniques, we provide a simple illustration of a model and an abstracted version of that model in Section 2.2. We then introduce the objective of this research effort, the motivation for performing the research, and how the end product benefits Rome Laboratory and the simulation community (Section 2.3). We conclude this section with a discussion of the foundations and the preceding work in which our work is rooted.

2.1 Concept of Model Abstraction

Since the focus of this effort is on model abstraction techniques, we begin by describing our concept of model abstraction. Figure 1 illustrates the modeling and simulation process. There exists some **real world system** (either actual or hypothesized), of which we want to create a model to use to answer some question(s). There is a broad range of the types of such questions, such as:

- How would a real-world system respond to particular stimuli?
- How should a real-world system be designed?
- Why is a real-world system behaving in a particular way (i.e., diagnosis)?
- How do people operate a real-world system (i.e., training)?

A model is used for a number of reasons, such as:

- Typically lower cost in using a model rather than the real-world system;
- Safety concerns in using a real-world system;
- Experimental control over the model;
- The non-availability of the real-world system (e.g., enemy weapon systems); and even
- The non-existence of the real-world system.

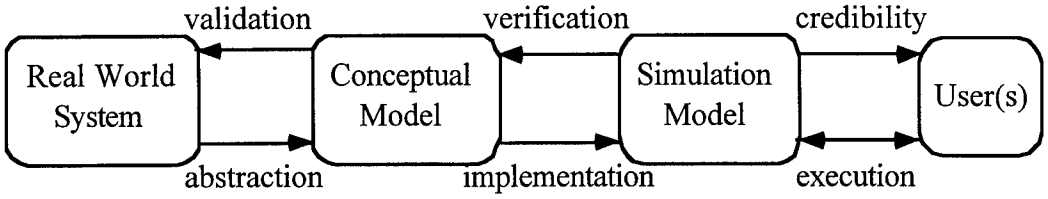


Figure 1, Model Abstraction in the Simulation Process

Development of a simulation model, as shown in Figure 1, involves two steps. The first step is development of a **conceptual model**, a way of "thinking about" and representing the real world system. Crucial decisions made by a model developer at this stage include:

- Determination of what factors influence system behavior;
- System behaviors to be incorporated into the model; and
- The manner in which system behaviors are represented.

This determination is based on factors including: the questions the model will be used to answer; the computational and development resources available to build, validate, and use the model; and the data available which describes the real-world system and its interfaces. We refer to this determination process as **abstraction**, since the conceptual model is a simpler representation of the more complex real-world system. The process of determining whether the conceptual model is an adequately accurate representation of the real-world system is **validation**.

A conceptual model is then **implemented** in the form of a computer-executable set of instructions known as the **simulation model**. **Verification** is the process of determining whether the implemented version of the model is an accurate representation of the conceptual model. The final phase of model development shown in Figure 1 involves the users interfacing with an executing simulation model. The extent to which the users believe that the simulation model is an accurate representation of the real-world system is defined as the **credibility** of the model. Of these steps in the simulation process, the focus of this study is on the first step, creating and manipulating the conceptual model.

Early work in the field of modeling and simulation focused on the development of single conceptual and simulation models for a real-world system. Such an approach works well if there is a small set of questions to be addressed using the model, and that set is known a priori before model development begins. However, such situations are rarely the case. New questions arise after the model is in use, models are decomposed and/or combined to address new questions, or the questions associated with a model cover a broad spectrum of perspectives of the real-world system. Zeigler (1984) introduced the concept of multifaceted modeling to provide a more effective approach to model development. A multifaceted model is a set of related conceptual models for a real-world system, emphasizing different aspects of the system required to address different questions about the real-world system. The term multifaceted recognizes the existence of multiplicity of objectives and models as a fact of life. It does not necessarily imply a hierarchical structure; rather, the concept supports partial decomposability of systems. This concept appears elsewhere in the literature as well. Falkenhainer and Forbus (1991) describe the idea of multifaceted models in terms of the ontological assumptions of model components.

A key to the multifaceted modeling concept is the relationship among the various conceptual models. We define a **model transformation** as a mapping from a conceptual model to a conceptual model. A transformation from a complex to a simpler conceptual model is a

simplifying transformation. A variety of simplifying transformations can be applied to a conceptual model. Our research is focused on those transformations which produce a conceptual model which requires less resources while still generating the same answers to questions about the real world system being modeled.

We thus define abstraction techniques as: *simplifying transformations that derive simpler conceptual models while maintaining the validity of the simulation results with respect to the question being addressed by the simulation.* Just as abstractions are a mapping from a real-world system to a conceptual model, they provide a mapping from one conceptual model to another conceptual model. The abstraction techniques discussed in this paper focus on mappings among conceptual models, although the principles can also be applied to development of conceptual models from real-world systems.

The validity of an abstraction is dependent on the question being asked. We refer to the question(s) for which the simulation is being executed as the **simulation requirement**. In C^3I modeling, the simulation requirement is typically some Measure of Effectiveness (MOE) of an entity, situation, process, or configuration being modeled.

This approach of defining abstraction validity with respect to a specific simulation requirement makes sense in the context of flexible, modular model structures which can easily be adapated to the simulation requirement. However, it is also useful to make some value judgements on the quality of a model abstraction independent of, or for an entire class of, simulation requirements. Abstractions should result in bounded deviations of model outcome, so that there are at least some model outcomes that will not change because of the application of a model abstraction. Second, the abstraction must result in a predictable deviation of model outcome. For example, a particular abstraction may result in always overestimating a particular output parameter. The result of the abstraction is predictable, in that it results in the value of that particular parameter increasing. We use the term **sound** to characterize an abstraction which is generally applicable to a broad set of simulation requirements. A sound abstraction is bounded or predictable.

The definition of abstraction techniques bounds our study in some significant ways. First, because we are dealing with the conceptual model, we do not include code optimization techniques in our investigation. Although abstractions change the implementation of a model to a simpler form, for this study we do not consider techniques that involve a more efficient implementation of the same conceptual model.

Model **refinement** is the opposite of model abstraction. It is the addition of complexity into a model to make it more representative of the real-world system being modeled, usually at the expense of additional computational requirements. We use refinement in the same sense that other researchers have used the term concretization. A transformation that maps a simpler model to a more complex model is a **refining transformation**.

We have defined model abstraction as a manipulation of the level of detail to which the real world system is represented. We refer to this level of detail as the **fidelity** or **resolution** of the model. We use these terms interchangeably, although we use fidelity when there is a connotation of the accuracy of the model representation. Resolution is used in situations where we wish to express the level of detail that is used to represent entities in the model.

In reviewing the literature, a more narrow definition of abstraction appears, as a specific technique that preserves exact simulation results, distinct from **approximation** (which generates inexact, but "good enough," simulation results). We take the liberty of maintaining our original definition of the term "abstraction", because of its expressiveness in the overall context of the model development process. We will use the term **aggregation** to refer to the more specific abstraction approach of removing detail in a model by combining elements of the model, such as system states.

In addition to the definitions provided above, we use commonly used simulation terms throughout this report. To ensure a common frame of reference for our terminology, we provide a set of definitions for these commonly used terms.

2.2 Illustration of a Model Abstraction Technique

To illustrate the definition of a model abstraction, consider the following simple example of a model of a grocery store adapted from Zeigler (1976). In the real-world system, there is a grocery store with one checkout lane. Customers enter the store, spend some time shopping, then get into the checkout line, wait (if necessary) until they reach the front of the line, then check out, and exit the store. The model input set consists of the time and identity of the customer when a customer enters the store. A shopping time duration is assigned to each entering customer from a probability distribution function. At the end of the shopping time duration, if the checkout line is empty (i.e., the checker is not busy), then a checkout duration time is assigned from a probability distribution function; at the end of the checkout duration time, the customer exits the store. If the checkout line is not empty, then the customer is inserted into a first in, first out queue. When the customer reaches the front of the queue, a checkout duration time is assigned from a probability distribution function; at the end of the checkout duration time, the customer exits the store.

The following informal description of the model is taken from Zeigler (1976).

Components

ENTRANCE, SHOP_AREA, CHECKOUT, EXIT

Descriptive Variables

1. Describing ENTRANCE

   HELLO—with range $\{\phi, a, b, c, \ldots\}$; HELLO = ϕ means no customer in ENTRANCE, HELLO = x means that customer named x is in the entrance.

2. Describing SHOP_AREA

SHOPPING_TIME—with range R^+; a random variable giving the elapsed time a customer will spend in the SHOP_AREA.

TIME_LEFT_LIST—with range $(\{a, b, \ldots\} \times R^+)^*$; $(x_1, \tau_1)\ (x_2, \tau_2)\ \ldots\ (x_n, \tau_n)$ means customer x_i will leave SHOP_AREA in time t_i from now.

Additional Terms and Definitions

State space (of a system): the collection of variables necessary to describe the relevant aspects of the entities of a system at a particular time. The state of the system at any given instant is composed of the states of the individual entities that make up the system.

Attribute: the state variables associated with each entity.

Discrete system: a system in which the state variable values change instantaneously at specific points in time.

Continuous system: a system in which the state variable values change continuously with the passage of time.

Hybrid system: a system that includes both discrete and continuous state variables. Most actual systems are hybrid systems, but usually one type of state change dominates. The systems typically modeled in C^3I applications, consisting of large collections of entities including military units, platforms, sensors, weapons, and so on, are such hybrid systems. The locations, velocities, and fuel levels of moving aircraft and ground vehicles are examples of continuous state variables, while the number of missiles remaining on a missile launcher is a discrete state variable.

Model: the term model covers a broad range of possible instantiations, ranging from conceptual models to physical analog models (see definitions of physical and mathematical models below). However, given the focus of this study, unless otherwise noted, our use of the term "model" refers to computer instantiations used for simulation purposes.

Physical models: models that are manipulated by direct physical analogy with the actual system. Examples include miniatures, cockpit simulators, etc.

Mathematical models: models that consist of logical and quantitative relationships that can be manipulated and changed to determine how the model reacts, and therefore how the actual system would react.

Deterministic simulation model: a simulation model that contains no probabilistic (i.e., random) components.

Stochastic simulation model: a simulation model in which the behavior of some state variables is represented as random values.

Continuous simulation model: a simulation model that is primarily based on differential or difference equations that define the relationships among continuous state variables.

Discrete simulation model: a simulation model in which the values of state variables change at specific points in (emulated) time. Discrete simulation models are frequently used to model continuous or hybrid systems, because the behavior of continuous state variables can often be approximated using various computational methods, such as interpolation. Typical C^3I models are discrete simulation models, in which some continuous state variables, such as aircraft locations, are approximated.

Event: the change in state variable value in a discrete simulation model. Since the simulation is generally characterized by these variable changes, discrete simulation models are often referred as discrete event simulation models.

3. Describing CHECKOUT

LINE—with range $\{a, b, ...\}^*$; LINE = $x_1, x_2, ... x_n$ means x_i is first in line, x_2 is second in line, ..., to be checked out (first in, first out queue).

SERVICE_TIME—with range R^+; a random variable assigning the time it will take for the customer who is now first in line to be processed.

SERVICE_TIME_LEFT—with range R^+; SERVICE_TIME_LEFT = σ means that the customer being processed will leave CHECKOUT in σ from now.

BUSY—with range {YES, NO}; indicates whether CHECKOUT is serving a customer.

4. Describing EXIT

BYE_BYE—with range $\{\phi, a, b, c, ...\}$; BYE_BYE = ϕ means no customer, BYE_BYE = x means customer x is leaving.

Component Interaction

A customer in ENTRANCE at clock time t is signaled by HELLO = x (customer's name); the customer immediately enters SHOP_AREA (HELLO becomes ϕ). SHOPPING_TIME is sampled for a value τ, and (x, τ) is then added to the TIME_LEFT_ LIST. As clock time advances, (x, τ) will be decremented until $(x, 0)$ is on the list. At this point customer x will leave the SHOP_AREA, immediately joining the back of the CHECKOUT_LINE. As prior customers in LINE are processed, the customer advances to the front of LINE. When the customer is first in LINE, SERVICE_TIME is sampled to get time σ, and SERVICE_TIME_LEFT is set to σ. Customer x waits at the head of the LINE until the SERVICE_TIME_LEFT becomes 0; then the customer departs, as signaled by BYE_BYE = x.

This simulation model could be used to determine the average amount of time a customer is in the store, or the average amount of time that a customer waited in line to check out. Suppose, however, that the simulation was to be used to determine the average amount of time that the checker was busy, and the average length of time the checker must work without a break. In this case, there are certain aspects of the model that are not relevant to the problem at hand. Specifically, since we do not care how long any individual customer is in line, we need not represent the individual customers in line. We can abstract the model in the following way. When a customer's shopping time is finished, the checkout time is immediately derived and added to a model variable that expresses the amount of time until the checker is no longer busy. The checkout line is eliminated (abstracted). The times when the checker's state (Busy or Idle)

changes are the only required model outputs. The queue, and the order of customers in the queue, is no longer relevant to the simulation requirement, and can be ignored.

The results of the abstraction expressed in terms of the preceding description are as follows: the EXIT component is eliminated, since the actual customer departure is not relevant; the other components remain. Among the descriptive variables, HELLO, SHOPPING_TIME, TIME_LEFT_LIST, SERVICE_TIME, and BUSY are not changed, while LINE, SERVICE_ TIME_LEFT, and BYE_BYE are eliminated. A new descriptive variable is added, defined as follows:

> CHECKER_BUSY_TIME_LEFT—with range R^+; CHECKER_BUSY_TIME_LEFT $= \sigma$ means that the checker will be busy until σ from now.

The component interactions for customers in ENTRANCE and SHOP_AREA are unchanged. When $(x, 0)$ is on the TIME_LEFT_LIST, customer x will leave the shop area. SERVICE_TIME is then sampled to get time σ, and σ is added to CHECKER_BUSY_ TIME_LEFT. (CHECKER_BUSY_TIME_LEFT is decremented as clock time advances.) When CHECKER_BUSY_TIME_LEFT reaches 0, BUSY is set to NO until another customer exits the SHOP_AREA.

The model output of checker state changes is identical for both the original model and the abstracted model. While there are outputs that the abstracted model cannot generate, they do not provide information that can be used to answer the question being asked.

2.3 Objective and Motivation

Having introduced the notion of model abstraction, we now establish the specific objective and motivation for this research effort. The objective of this research effort is to identify, categorize, develop and demonstrate model abstraction techniques for manipulating software simulation models and/or model components. These techniques are used to decrease the model fidelity to match the requirements of the simulation and/or the computational resources available for executing the model. The concept of manipulating simulation model fidelity to accommodate simulation requirements and/or computational resources is not new; it is generally an integral part of the simulation development process. However, this aspect of simulation development has generally been performed in an ad hoc manner. Recent work in developing models with components of varying resolution and fidelity, and in reusing model components, has renewed interest in effective means of manipulating model fidelity while preserving the validity of results. Developing such techniques is the focus of this research effort.

A better understanding of model abstraction techniques is also important in designing and using models which include processes and entities at distinctly different levels of resolution. Model abstraction techniques are also required to realize the concept of selective fidelity in models, in which the level of model resolution is tuned to meet the requirements of the simulation. Thus the motivation for this effort is to provide model developers with a set of tools

that can improve the modeling and simulation process at three points: design, execution, and reuse. We address each of these areas in a subsection below.

2.3.1 Improving Model Design

Understanding of model abstraction techniques and their application can provide the model developer with a better tool box for conducting model design. By drawing on specific techniques from other disciplines such as qualitative reasoning, we provide the model designer with a broader catalog of techniques to apply to the problem of designing a model at the appropriate level of resolution and fidelity. Also, by understanding the interactions and interrelationships among abstraction techniques, the model designer can avoid problems that arise from a mismatch of levels of fidelity. Examples of such mismatches include one entity being modeled in too much detail relative to other entities in the model, and an inappropriate choice of time resolution for the behaviors being represented in the model.

Ultimately, greater understanding of model abstraction techniques is part of an overall maturing of the model development discipline from a craft to a more engineering oriented discipline. Model development as a craft depends on the experience of the individual model developer. In maturing to an engineering oriented discipline, much greater direction is provided to the model developer in making crucial model design decisions.

2.3.2 Improving Model Execution

Perhaps the most obvious motivation for investigating model abstraction techniques is the potential reduction in resource requirements for simpler models. Model development has always involved a tradeoff between the complexity of the model and the correlated resources required to design, execute, and validate the model, and the accuracy and precision of the model. Abstractions move along the tradeoff curve in the direction of reduced resource requirements. While one typically thinks of computational resources (such as compute cycles) in this context, the level of abstraction of the conceptual model impacts a number of other resources. Examples of such resources include (but are not limited to) the following:

- Development time to design, code, and test a simulation model;
- Time required to document a model;
- Execution resources (such as CPU cycles and memory);
- Model storage requirements (e.g., disk space);
- Time required to collect data for model validation; and
- Time required to validate the model.

Clearly developing simpler models has a potentially significant payoff.

In addition to resource savings in model execution, abstraction techniques are also motivated by the evolving concept of selective fidelity. The idea of selective fidelity is to only model in detail the behaviors of the real-world system that are necessary. Selective fidelity can involve fidelity decisions at "compile time" or "run time". Compile time selective fidelity requires the capability to select the appropriate model for a particular application, without requiring significant changes to any of the interfaces that the model has with external elements.

"Run time" fidelity decisions, which we refer to as dynamic selective fidelity, are characterized by selection of a model or model component(s) for execution based on the current value of the system state. For example, consider a model of ground unit activity and an airborne sensor. A detailed model is only required when the unit is within the sensor's active footprint; units outside the sensor footprint at that simulation time need only be modeled at a coarse level. However, as the sensor footprint moves (which could be dictated by events external to the simulation, such as operator input), the set of units that must be modeled in detail changes. Model abstraction techniques are the key to achieving the range of selective fidelity, including dynamic selective fidelity.

2.3.3 Improving Model Reuse

Model abstraction techniques also have potential benefits for enabling and improving model reuse (Frantz 1994). One area gaining increasing popularity in terms of model utilization and reuse is integration of models of different levels of resolution. The integrated model provides capabilities that are generally difficult to achieve by expanding individual models. For example, integration of an air-to-air combat model with an air campaign model provides a more accurate resolution of air combat within the air campaign model. In addition, it provides a broader scope and more realistic context for the air-to-air combat model. Taken together, the integrated model provides a broader scope and detail than either individual model (assuming the simulation requirement is for a campaign level model with representation of individual combat).

The value of established model abstraction techniques is that they provide a means to determine if one model is a valid abstraction of the other, and if not, the modifications that are required to make one model a valid abstraction of the other. Often such integrations are implemented without addressing this issue explicitly, or the final product requires a new validation procedure. Well-defined model abstraction techniques can be used to enhance the validity (and facilitate the validation) of the results of such integrations.

Model abstraction techniques also can facilitate model reuse, since they provide a means for transforming models to meet simulation requirements. They also provide a step towards measuring the level of resolution of a model, which could be a useful metric in determining which of several candidate models is most appropriate for a reuse application.

2.4 Foundations

There is a considerable body of work on which this effort relies. As we have already indicated, neither the notion of model abstraction nor definition of the techniques themselves is new. The concept of model abstractions shows in up in a variety of disciplines. For example, Simon and Ando (1961) and Ijiri (1970) describe abstractions of economic models. Zeigler and Weinberg (1970) describe abstractions applied to biology models. Agrawal (1985) compiled a compendium of articles on abstractions in analytic queuing models. The common denominator in all these studies is the simplification of the model to be more mathematically tractable.

Our primary emphasis in this effort is to catalog various techniques and demonstrate their applicability to the general domain of C^3I modeling. In this section, we briefly summarize the key foundations on which this work rests. More specific discussion of this work is the focus of subsequent sections of this report.

The first area from which we draw is that of traditional discrete event simulation. Techniques for model simplification and abstraction have been in existence since the first model was coded. In previous generations of computer systems, in which memory and compute power were typically more constrained, decisions about model complexity were the highest priority in the design process. Real-time simulators such as flight simulators also demanded that particular attention be paid to the resolution and fidelity of the models. Even the concept of dynamic selective fidelity has been addressed in the implementation of visual systems for real-time simulators. Using eye tracking devices, the fidelity of the generated scene where the eye is pointed is greater than the resolution of the surrounding scene, which is only in the humans' peripheral vision. We observe that such techniques have generally been applied in an ad hoc fashion, without a comprehensive understanding of the available techniques, their implications, or the relationship between particular types of models and simulations and abstraction techniques.

The first attempt at formalizing discrete event simulation and thereby providing a formal basis for definition and discussion of abstraction techniques, was presented by Zeigler (1976). Zeigler includes an informal discussion of abstraction techniques, then provides a formal basis, called the Discrete Event System Specification (DEVS), for reasoning about discrete event simulation systems. We summarize that formalism in Section 3.3. Other formalisms have been advanced since then by Ho (1989) and Radiya and Sargent (1994). For our purposes, we use the DEVS formalism to assist in the organization of techniques in the taxonomy presented in Section 5 of this paper. A significant exercise, but beyond the scope of this effort, is to define the various abstraction techniques in a formalism such as those cited above and prove the behavior preservation properties (See Section 11.2.1).[1]

[1] The focus of this effort is to demonstrate applicability to the customer simulation and modeling domain. For this reason, we moved on to demonstrating techniques even though they are not formally proven. These techniques can still be applied, relying on traditional validation procedures to establish the validity of the results. The value of a more formal approach is the efficiency of validating a process rather than validating each product of the process.

The other domain of previous research from which we draw is that of model-based reasoning, a field of study in artificial intelligence (AI). AI researchers, beginning in the early 1980s, attempted to develop systems that would automatically reason about common physical (i.e., real world) systems. Researchers observed that humans qualitatively reason about such things. For example, we do not know the capacity of a bathtub, or the flow rate of water into a bathtub, but we do know that if the faucet is left on and the drain is closed, the bathtub will eventually overflow. Thus much research was focused on qualitative, rather than quantitative, models of real world systems. These models tend to generate complete envisionments (all possible state space trajectories), which is theoretically feasible in a qualitative model having a finite number of qualitative states. However, reasoning about real world situations requires models of sufficient complexity that generation of envisionments rapidly becomes computationally demanding. The next step was to develop a hierarchy of models with varying degrees of complexity, and execute the simplest model required for the reasoning to be performed (i.e., that met the simulation requirement). At this level this is precisely the problem we are attempting to address, and thus we have spent considerable effort in reviewing abstraction techniques from the field of qualitative reasoning for our effort.

Fishwick and Zeigler (1992) note substantial parallels between their work and the ongoing research in qualitative simulation. Fishwick and Zeigler's argument is generally that the qualitative simulation community has failed to build on the systems theory that addresses some of their specific issues. They also caution that qualitative simulation should be carefully applied, since quantitative simulation can address many of the issues that motivate qualitative simulation. From our perspective, our review of the qualitative simulation research is not for the techniques of qualitative simulation, but rather the specific issue of multilevel models and abstraction techniques.

This realization of the potential synergism of qualitative and traditional simulation techniques appears as a frequent theme of several authors. Fishwick and Zeigler (1992), Miller, Firby, Fishwick, Franke, and Rothenberg (1992), Fishwick (1992), and Fishwick, Narayanan, Sticklen, and Bonarini (1994) provide general rationale and approaches for synergizing traditional simulation and AI modeling and simulation techniques. The remainder of this report is an exploration of a specific focus of that synergy: model abstraction techniques.

3. Abstraction Techniques from Discrete Event Simulation

There is a substantial baseline of existing work in the application of model abstraction techniques to modeling and simulation. In this section, we briefly summarize the ideas of two key researchers in this field. We begin with a summary of the abstraction approaches advocated by Zeigler (1976) that motivated the development of the DEVS formalism. Section 3.2 includes a presentation of an earlier taxonomy of abstraction techniques developed by Fishwick (1988). We conclude this section with a discussion of a specific formal approach to discrete event simulation. Beyond Zeigler and Fishwick, much of what we can draw upon from the discrete event simulation community is in the form of anecdotal experience, rather than well established scientific foundations for model abstractions and model fidelity manipulation. Most standard simulation textbooks do not address the subject, yet we suspect that almost every simulation practioner, at one time or another, was faced with the decision of how to reduce the resource requirements of a simulation without compromising the quality of the simulation results. As a result, the authors' own experience and observations are reflected in the taxonomy in Section 5 as well as the formal ideas of Ziegler and Fishwick.

3.1 Model Abstraction Techniques: Zeigler

Zeigler (1976) devoted a significant portion of his 1976 text on modeling and simulation to developing a formal framework for describing discrete event simulation, including operations of model abstraction. He discusses model simplification (which equates to simplifying transformations in our terminology). Zeigler uses the term "base model" to express the most detailed model, and the term "lumped model" to express the results of simplifying transformations applied to a model. The use of the term "lumped model" suggests abstraction by aggregation; however, as we define aggregation, the simplification techniques described by Zeigler actually include abstractions by means other than aggregation. He identifies four general categories of techniques:

1. Dropping of one or more components, descriptive variables and/or interaction rules.
2. Replacing one or more deterministically controlled variables by random variables.
3. Coarsening the range sets of one or more descriptive variables.
4. Grouping components together in blocks and aggregating the descriptive variables within blocks.

The following paragraphs summarize Zeigler's approach in each of these areas.

Dropping Components: The factors that are included in a model can have a varying influence on the model outcome. Some factors have a major contribution, while others can have a minor contribution. Reducing the number of factors can simplify models. Reduction is based on eliminating factors which least affect model behavior within the experimental frame of interest.

This concept is equivalent to the concept of finding the appropriate system boundary (see Section 4.4.2).

By eliminating a factor, the abstracted model provides an approximate but valid result, which is also the objective of Weld's use of model sensitivity analysis (see Section 4.4.3). Zeigler describes this technique as similar to an engineering approximation, but does not specifically identify his approach as abstraction by approximation. He states that the abstracted model can only be validated by comparison with real data. Weld's approach is based on finding a fitting parameter such that the approximate result has a bounded variance from the more detailed model, thereby removing the requirement for real data to validate the abstraction.

There is potentially a relationship between entities, descriptive variables, and interaction rules, such that dropping an entity may require dropping associated descriptive variables and interaction rules, and so on. The dropping of associated descriptive variables is an instance of a more general set of abstractions that are triggered by some initial abstraction. While Zeigler mentions this relationship in passing, we will return to this concept in Section 10.2.

Replacing Deterministic with Random Variables: This technique of model abstraction involves replacing some set of interaction rules that determine model output by deterministic computation with a (simpler) interaction rule based on a probability mechanism.

Coarsening Range Sets: In this simplification procedure, the descriptive variables in both the base model and the lumped model are the same, but the range set of the variables is smaller than the range set for the variables in the base model. One example of coarsening is rounding off variable values. We return to this idea in Section 4.3, where we draw the same conclusion from a much different perspective, that of a continuum of qualitative-quantitative modeling. "Quantizing the model" is another way to express the type of abstraction, where a range of model variable values is mapped to a single simulation state.

Another example is termed "classification". Zeigler explains this simplification technique with an example of a model of an elevator. The people in the elevator could be represented as individuals with names, or more coarsely as an integer number of people in the elevator, or even more coarsely as a Boolean value indicating whether or not there are people in the elevator.

Grouping Components: Components can be combined or grouped to abstract a model. The examples provided by Zeigler are generally hierarchies in which the lower level components are controlled by the higher components is some way. We will later observe that this is one of many criteria by which components can be grouped.

3.2 Process Abstraction: Fishwick

Paul Fishwick is another researcher from the simulation community who has investigated the issues of levels of resolution and abstraction. Fishwick (1988) defines an approach to abstraction based on processes represented in a model. The justification for the use of abstraction includes the following rationale:

1. An abstract model is usually less computationally complex than the base model.
2. An abstract model is easier to understand in most cases.
3. The creation of abstract models facilitates construction of a library of different models that represent the same process (which he describes as a "network").
4. In many cases the library of models for one process represents an evolutionary path for the modeling process. Processes are often modeled using simple methods at first, and then evolve with added detail to become more complex representations as the development cycle continues.

An abstraction network is defined as $A = [S,\varphi]$, where:

S is the set of systems or models representing the same process at different levels of abstraction.

φ is the set of possible abstraction relations and induces a partial ordering over S. i.e., $\varphi = \{ \rho_i \}$ where any given ρ_i represents a method of abstraction

Fishwick defines a system S_i such that $S_i \in C_m$, where C is the set of components for a system. A system is denoted $S_i = < c_{1m}, c_{2m}, ... c_{im} >$ as an m-valued system specified at abstraction level i. In other words, a system is a set of inputs, a set of outputs, and a set of valid relations.

Fishwick describes seven forms of process abstraction:

1. Abstraction by Representation: the abstraction S_i "represents" S_{i+1} in another form.
2. Abstraction by Induction: an abstraction by which a set of behaviors is "coalesced" into a single model representation. For example, a repeated binary series could be represented by a single definition along with a statement of the number of repetitions.
3. Abstraction by Reduction: an abstraction based on reduction in the set theory sense.
4. Total System Morphism: is a complete mapping between all $c_{(i+1)j}$ to c_{ij} in S_{i+1} and S_i respectively for a structured system specification. A total system morphism preserves features of the system. It is ideal for representing abstractions on systems defined as discrete structures, such as directed graphs, but less useful for systems defined as continuous functions and systems.
5. Partial Systems Morphism: defined as a set of morphisms between some components, but all relations and functions are not necessarily preserved during mapping. Fishwick notes that his definition seems "very open ended" but it is "included within the taxonomy for the sake of completeness especially when we

consider certain graph algorithms such as those to determine strongly connected components in an arbitrary directed graph, interval analysis, and other topics..."

6. Sensory Abstraction: abstraction by computer graphics. The example cited in the article is that of dynamic models of fire by particle systems. It provides "the viewer with realistic sequences of both fire and explosions over time." "The important note is that the particle system approach represents an abstract modeling approach to a physical system that is highly complex ..." In the context of computer graphics and perception of motion, the model need not actually model reality. It need only generate behavior that makes it appear to do so to the observer. This abstraction is a black box approach to abstraction, in that it ignores the internal structure of the system and merely portrays the external observables.
7. Cerebral Abstraction: defined as those abstraction methods that deal with modeling intuitive processes at high levels of abstraction. Two methods are identified. The first, system dynamics, occurs when causal graphs are constructed with feedback loops. The second method, qualitative reasoning and simulation, is where a central theme is to provide some means of: creating an abstract model; applying an analysis procedure to the model to create a causal model; and providing an explanation feature to describe the nature of the device.

To illustrate these various abstractions, Fishwick provides an example abstraction network using the "dining philosophers" as a sample physical process.

1. He provides a DATA table of states: (1,0,1,0,0) for time t=0
   (0,1,0,1,0) for time t=1
   and so on for each time increment,

   where $n_i = 1$ signifies that the i_{th} philosopher is eating.
2. He then shows that an abstraction by induction is possible that replaces the table by a Finite State Automaton (FSA).
3. He also expresses the same state transitions in a Petri Net, and then shows how Partial or Total Systems Morphisms (TSM) can be applied to transform that into a second simpler abstracted Petri Net.
4. With the assumption that PNET1 is synchronous (denoted by behavior β_s) he states that the system morphism $\beta_s(\text{PNET1}) \rightarrow \text{DATA}$, $\rho = \text{TSM}$ is valid.

3.3 A Formal Framework for Discrete Event Modeling

Recognizing that concepts such as those summarized in the preceding sections were somewhat informal and ad hoc, Zeigler drew on the area of systems theory to define a framework for modeling and simulation. The framework consists of three primary objects: the real system, the model, and the simulator, as shown in Figure 2. The model and the real world are related by a modeling relation, which defines the extent to which data generated by the model agrees with data produced by the real system. The simulation relation represents the extent to which the simulation executes the set of instructions that comprise the model.

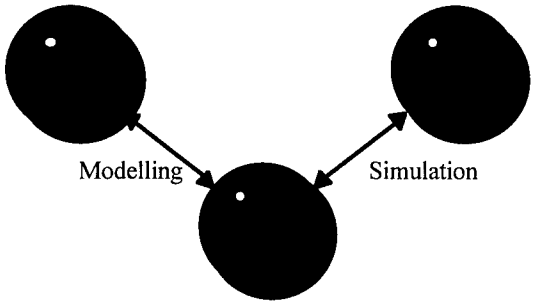


Figure 2, Basic Elements of Simulation and Modeling (Zeigler 1976)

These relationships are expressed in terms of a formalism called the Discrete Event System Specification (DEVS) formalism. The DEVS formalism provides a means for defining the structure of a model mathematically. This formalism is then used to define various levels of system specification, which are characterized by their level of abstraction, and preservation relations between specifications, called morphisms. Morphisms can be used to relate models in different formalisms at the same level of specification, or models at different levels of specification. The morphisms provide a means of relating models at various levels of abstraction. Such relationships can then be used to facilitate the organization of model components in such a way that models can be composed and manipulated. The System Entity Structure (SES) provides the means for describing components and their interactions.

A synopsis of DEVS and the SES is provided in Appendix A.

4. Abstraction Techniques from Qualitative Reasoning

One research area in which model abstractions have been extensively considered is the qualitative reasoning/qualitative simulation field of artificial intelligence. The purpose of this section is to summarize the abstraction techniques used in qualitative reasoning and qualitative simulation, focusing on the techniques used to relate various models (of differing fidelity) of the same physical entity or process. Where applicable, we relate specific notions and concepts in this domain to comparable concepts in other domains.

This section is divided into four subsections. We begin (in Section 4.1) with a brief discussion of qualitative reasoning and qualitative simulation, its origins, and the rationale for consideration under the Model Abstraction Techniques effort. This introduction is intended to identify key concepts that are discussed in subsequent sections; references are included to sources that provide a broader and more thorough discussion of qualitative reasoning and qualitative simulation techniques. We then describe a number of different approaches documented in the literature; three sections are included, each discussing a particular technique or category of techniques, as follows:

- Behavior abstractions (Section 4.2);
- Quantitative extensions to qualitative simulation (Section 4.3); and
- Model abstractions (Section 4.4).

We conclude this section with some observations about similarities and differences between qualitative and discrete event simulation.

4.1 Qualitative Reasoning Background

This section focuses on research work being conducted in a field of artificial intelligence called qualitative reasoning. The motivation for qualitative reasoning research is to create systems which reason about physical systems that are encountered in everyday life. There are many situations in which humans do not know the specific quantitative information about a physical system, yet are able to draw reasonable conclusions about the behavior of that system. For example, if one fills a bucket with water, then finds it half empty a few minutes later, one can conclude that there is a leak in the bucket. It is not necessary to know how much water was originally in the bucket, or how much water remains, or how fast the water is running out of the bucket. Further, one can predict that the water will continue to run out until the water level is below the location of the leak. The reasoning is based not on computation of specific quantities involved, but rather on the relationships of the quantities. We conclude that there is a leak because the amount of water at time t_1 is less than the amount of water at time t_0. Furthermore, based on our previous (qualitative) experience of containers of liquid, we can predict that the water will stop running out at some future point, even though we do not know how much water will remain at that point, nor how long until the water will stop flowing out of the bucket.

The capability to perform this type of reasoning is at the heart of qualitative reasoning. People use qualitative descriptions of physical systems that capture distinctions that make an important, qualitative difference, while ignoring other distinctions. These descriptions are one or more models of the physical systems. These models typically reflect differing perspectives of the system. Models vary in the factors that they represent to explain system behavior (e.g., different models embody different abstractions of the real-world system being modeled). Our particular interest is in how models are selected and composed, and how models of differing resolution are related. A number of AI researchers have been investigating and developing techniques to formalize this reasoning approach. There are several approaches to qualitative reasoning, such as qualitative simulation (Kuipers 1986), confluences (de Kleer and Brown 1984), and Qualitative Process Theory (Forbus 1984). Key concepts of each approach to qualitative reasoning are addressed in a subsection below.

4.1.1 Qualitative Simulation

Qualitative simulation has evolved over the past decade as an integral tool for supporting qualitative reasoning, while exhibiting significant analogs to traditional simulation. Thus we begin our discussion of qualitative reasoning techniques with the topic of qualitative simulation.

A system to perform qualitative reasoning is generally constructed from models of the systems that are the subjects of the reasoning. Kuipers (1994) defines a model as "a (small) finite description of an infinitely complex reality, constructed for the purpose of answering particular questions." Reasoning with models then involves two steps. First, one must select an appropriate description of the system, i.e., the model. Then a description of dynamic system behavior based on the model must be generated. We refer to this process of generating a dynamic description of system behavior as a simulation. Executing a simulation makes explicit facts about the world that are implicit within the description of the system (the model). These models are recognized as abstractions of real-world systems; Kuipers (1994), for example, defines differential equations as an abstraction of a real-world physical system, and a qualitative model as an abstraction of differential equations.

This concept of a model is completely consistent with the concept of a model within the discrete event simulation domain. In both domains, models are considered abstractions of real-world systems and form the basis for a dynamic description of system behavior generated by a simulation. Furthermore, as in the case of discrete event simulation, it may be necessary in qualitative reasoning to represent the same physical system or process with multiple models, depending on the particular issue to be addressed using the simulation. For example, Kuipers (1994) notes the importance of using multiple models describing the same physical system to answer a question about the system. The key to efficient problem solving using simulation is selecting the proper model for addressing the issue at hand. It is these similarities between simulation supporting qualitative reasoning and discrete event simulation that we want to exploit to enhance model abstraction techniques in the discrete event simulation domain.

A qualitative simulation model is a set of qualitative differential equations (QDEs), which are abstractions of ordinary differential equations that describe the relationship of variable values of a representation of a system. A QDE includes a set of variables, a quantity space for each variable, a set of constraints applying to the variables, and a set of transitions that define the boundaries of the domain of applicability of the QDE. Rather than the domain of real numbers (the typical quantity space for variables in a discrete event simulation model), the quantity space for a variable in a qualitative simulation model is defined by an ordered set of **landmark values**. Landmark values include negative infinity, zero, positive infinity, and values with specific meaning for the model. The value of a qualitative variable includes its value relative to the ordered set of landmark values and its rate of change. Qualitative constraints describe constraints on the relationships among qualitative variables, and include relationships such as addition, monotonically increasing or decreasing functions, and rate of change.

A qualitative simulation problem specifies a QDE as a set of variables, their quantity spaces, and their constraints with corresponding values. The problem is then to determine possible behaviors given a set of initial conditions. The dynamic representation of system behavior is generated by determining successor states to the initial state that satisfy the constraints. Alternative approaches can be followed to represent the sequence of states. A behavior tree can be generated, starting with the initial state and generating all possible sequences of successor states along the branches of the tree. Alternatively, all possible states can be generated, followed by an analysis of the valid state-to-state transitions. This form of the simulation is represented as a transition-graph that shows the successors to each state. The set of all possible states and state transitions is called the **total envisionment** of a QDE. The states reachable from a specific initial state is referred to as the **attainable envisionment**.

Time is handled as an alternating sequence of time intervals and specific points in time where qualitative changes to variables occur. The initial state represents conditions at the specific time-point t_0, which is followed by the open time interval (t_0, t_1), where t_1 is a time-point when a qualitative change to the simulation occurs.

An algorithm for generating behavior trees, given a qualitative model and initial conditions, has been implemented. That implementation is referred to as QSIM.

4.1.2 Confluences

Another approach to qualitative reasoning is described by de Kleer and Brown (1984). Much like qualitative simulation, de Kleer and Brown develop an approach to describing the behavior of physical systems in terms of qualitative parameters. The parameters are defined in terms of a quantity space similar to that defined by Forbus (1984) for Qualitative Process Theory. De Kleer and Brown's quantity space is simpler, and is local for each component rather than global for the entire device model.

The specific objective of de Kleer and Brown is to infer behavior of physical devices from descriptions of their physical structure. The physical structure is described in terms of

components and interconnections among components. The components are described in a library of generic component descriptions. These component descriptions are expressed as a set of confluence equations, or confluences. These confluences are similar in concept to qualitative differential equations. Using a combination of constraint propagation and generate and test, all solutions to the confluence equations are generated.

The result of the confluence solution is an envisionment, which shows all behavior sequences under all possible initial conditions. Given initial conditions, this state diagram can be used to trace state trajectories. For this reason, the approach described by de Kleer and Brown (1984) is less similar to the notions of traditional simulation than qualitative simulation described in Section 4.1.1. However, the concept of envisionment is useful in considering state aggregation techniques described later in this section. Also, the abstraction of real world physical systems to confluences is very similar to that used in qualitative simulation.

4.1.3 Qualitative Process Theory

Qualitative Process (QP) Theory (Forbus 1984) was developed during the same time frame as qualitative simulation. It is a tool for analyzing behavior of physical systems based on the qualitative relationships of the parameters which describe the systems. QP Theory is based on an ontological perspective that the real world is made up of: physical objects, which have continuous properties; and physical processes, which change the properties of objects. Forbus and de Kleer (1993) note that this ontological perspective is different from component-based ontologies such as those described in the preceding sections.

Despite the different ontological perspective, QP Theory shares significant fundamental concepts with qualitative simulation and confluences, such as quantity space (a collection of ordinal information relating a parameter to other significant parameters) and landmark values. Equations relating parameters are constructed from qualitative proportionalities. For example, the statement that "Q1 is qualitatively proportional to Q2" means that:

1. There exists some function which determines Q1;
2. The function depends at least on Q2; and
3. The function is increasing monotonic in its dependence on Q2.

Specific values of variables can be defined at a finite number of points using correspondences. In addition to qualitative proportionalities, QP Theory also includes direct influences. Direct influences express differential relationships, where constraints are placed on the derivative of a quantity rather than the quantity itself. Thus Q1 directly influences Q2 if:

$$\frac{dQ_1}{dt} = \ldots + Q2 + \ldots$$

Note that the sign preceding Q2 can be positive or negative.

The combination of qualitative proportionalities and direct influences provides a means of expressing qualitative versions of ordinary differential equations. A qualitative equation serves as both an abstraction of a set of quantitative equations, and as a definition of causality relationships among parameters. Note that a single qualitative proportionality or direct influence does not necessarily define all of the influences that determine the value of a parameter. These can be combined using influence resolution to determine the effect. Restrictions on allowable combinations of equations ensure that the causal relationships are clear. For example, direct influences can only be imposed by physical processes. Also, if the quantities and qualitative proportionalities are considered as the nodes and edges of a graph, respectively, the graph must be free of cycles. Finally, no quantity can be directly and indirectly influenced at the same time.

Much of the basic conceptual foundation of QP Theory is shared by qualitative simulation. As the ideas have evolved, research building on QP Theory has tended to focus more on the model development aspect of the problem, while the focus of the QSIM work has been on model execution and simulation. We discuss in more detail some relevant concepts in QP Theory model construction in Section 4.4.1.2.

4.2 Behavior Abstractions within a Qualitative Model

With the background on qualitative reasoning established, we now turn to the specific issue of model abstractions within a qualitative reasoning system. The first types of abstractions that we consider are abstractions of behavior within a qualitative model. For many complex dynamic systems, there are distinctions among possible behaviors of the system that make the simulation intractable, and yet are of no interest to the modeler. These distinctions clearly add unnecessary complexity to the model. Within the context of qualitative simulation, the objective of the behavior abstraction techniques is to combine simulated behaviors whose differences are irrelevant to achieving the simulation objectives. A simulation objective can be a query that the qualitative reasoning system poses to the model; this equates to our concept of a simulation requirement.

The behavior abstraction techniques identified in the qualitative simulation literature include the following:

- Behavior aggregation;
- Chatter box abstraction;
- Model decomposition;
- Time scale abstraction;
- Other aggregations; and
- Aggregation of repeating cycles.

Each of these techniques is described in a subsection that follows.

4.2.1 Behavior Aggregation

The concept of behavior aggregation is based on the fact that qualitative simulation can be used to determine all (or all possible) simulation states. There are two principal representations for possible behaviors of the system, as described by Kuipers (1994). One is the behavior tree, in which successor states are determined successively for each state beginning with the initial state. The other is the total envisionment, in which all possible states of the system are generated, and an immediate-successor relation is used to link the states into a transition graph. All possible behaviors are then implied by paths through the graph.

One extension to the initial qualitative simulation work is described by Clancy and Kuipers (1993) and involves techniques for aggregating behaviors. It is based on characterizing all possible behaviors of the system (i.e., all sequences of system states) as a lattice. This lattice of finite descriptions highlights different distinctions at various levels of detail. The different levels of detail allow the modeler to trade off the computational requirements (and ultimately the tractability) of the model and the level of detail required to perform the reasoning (i.e., the simulation requirements). Abstractions ensure that only those distinctions that are relevant to the simulation requirement are computed.

States are aggregated to eliminate occurrence branching, which is due to the complete temporal ordering of events whose order is not constrained by the qualitative differential equations that describe the system. If the temporal order of the events is irrelevant, such that either branch of behaviors leads to the same subsequent behavior, then the distinction between the two possible sequencing of events is irrelevant, and the two distinct paths can be aggregated into a single path. In other words, Single Input Single Output (SISO) subgraphs of an envisionment graph of the simulation are aggregated into a single aggregate state. Within this aggregate state, temporal correlation between events is eliminated, and the simulation is continued for this subtree from a single abstract state representing the common properties.

4.2.2 Chatter Box Abstraction

Another source of irrelevant distinctions in the behaviors of qualitative simulations occurs because of chatter. Chatter occurs when the derivative of a variable is constrained only by continuity; that is, the direction of its value could be any qualitative value. For example, chatter could occur when a variable is defined as the difference between two variables that are monotonically increasing functions. The simulation branches on all possible values for the qualitative derivatives of the chattering variables, resulting in intractable branching. Chatter box abstraction, described by Clancy and Kuipers (1993), eliminates chatter by abstracting the chattering region into a single qualitative state. A chatter box is a region within the state space of the model in which the qualitative derivatives of potentially chattering variables are allowed to vary, while the qualitative values of the other variables remain the same.

4.2.3 Model Decomposition

The application of model decomposition within the qualitative simulation domain is also motivated by the level of complexity of real world system models and the requirement to abstract models for tractability. As with other abstraction techniques used in qualitative simulation, this technique eliminates model computations that involve details that are irrelevant to the question at hand. The remainder of this section is taken from Clancy and Kuipers (1994).

The philosophy of model decomposition is to divide the model into loosely connected components, then model each component separately while specifically addressing the component interactions. Model decomposition uses a causal analysis of the constraints in the model to partition the variables into tightly connected components. The causal relations among components are analyzed such that execution of one component can constrain the behavior of subsequently executed components. Also, where there is no causal relationship, the components can be executed in parallel.

The model decomposition algorithm involves three major steps: variable partitioning, creation of submodels, and then executing the partitioned simulation. The objective of variable partitioning is to combine variables that are tightly connected within the constraint network, and separate variables that are not connected. Currently, manual partitioning techniques based on the concepts of Iwasaki and Simon (1986) and Iwasaki (1988) are used to partition the variables.

The second step involves creating sub-models based on the variable partitioning. The sub-models can contain two types of variables: within-partition variables that belong to the partition, and boundary variables. Boundary variables are non-partition variables that are related to a within-partition variable through a constraint and are causally upstream or acausally related to the variable.

A separate QSIM simulation is performed for each sub-model, generating a state-based behavioral description for the within-partition variables of the sub-model. The description of a boundary variable's behavior generated by an upstream sub-model is used to guide the simulation of a down-stream component.

Since each component sub-model is only responsible for generating behaviors of the within-partition variables, distinctions in behaviors of boundary variables are not relevant. Abstract states are created to encompass the system states which "fundamentally" differ only in the value of the boundary variables.

4.2.4 Time Scale Abstraction

Another technique developed to address the tractability of qualitative simulations is that of time scale abstraction. The notion of representing and analyzing systems whose processes vary widely in time frame has been considered for many years. Simon and Ando (1961) present an early example of examining time scales to assist in making system representations more tractable.

In this section, we review approaches to time-scale abstraction that have been developed in the context of qualitative simulation.

4.2.4.1 Qualitative Simulation Time-Scale Abstraction

Kuipers (1994) defines the basic principle of time scale abstraction as follows: "If a complex system can be decomposed into mechanisms that operate at widely separated time-scales, then a particular mechanism can view a faster one as being instantaneous, and a slower one as being constant." There are a number of keys to implementing time-scale abstraction:

1. Developing a hierarchical structure of model components that maintains consistent representations of the entities in widely varying time scales;
2. Defining the mechanism to shift control across mechanisms within the hierarchy;
3. Communicating information across levels; and
4. Defining what constitutes a sufficiently wide separation of time-scales.

The hierarchical structure in qualitative simulation is based on treating the equilibrium state in a faster model as the initial conditions for a slower model, and propagating the final conditions of a slower model as a new equilibrium state of the faster model. In order to maintain consistency of the results, there are several assumptions made about the faster and slower models. First, variables with the same names in the models are assumed to mean the same thing. Second, all independent variables in the faster model must appear as variables in the slower model as well. In addition, the concept of corresponding states (states that describe the same situation) is defined. A variable that appears in corresponding states must denote the same value, representing corresponding landmarks. If a variable appears in both models, and a landmark of the variable in one model corresponds to an open interval between landmark values in the other model, a new landmark value can be created in the second model to correspond to the landmark value in the first model.

In order to translate a monotonic function from a faster model to a slower model, it is necessary to first determine which abstracted monotonic functions are used in the slower model, since there may be several abstractions of the faster model (not all of which are relevant to the slower model). Then the abstracted function constraint in the faster model must match the constraint used in the slower model. Once the constraint has been established, the corresponding value tuples from the faster model must be mapped to the slower model.

The mechanism used to shift control from the faster model to a slower model is to abstract the fast process to a multivariate monotonic function constraint.

4.2.4.2 Temporal Aggregation

Another perspective of abstraction of the time element of a model comes from Iwasaki (1992). Temporal abstraction is described as ignoring behavior over a short period of time. Temporal abstraction is based on a comparison of model temporal grain sizes that results in one

of three outcomes. A model is defined as slow if its grain size is much larger, and generates detectable effects in the time period of interest. A model is defined as fast if its grain size is much smaller, in which case its processes may appear as discontinuous changes to the slower model. The third outcome is approximate equality, i.e., neither slow nor fast by these definitions. This approach is part of an iterative approach to selecting model components. Iwasaki advocates building separate models where temporal grain sizes of components are not roughly equal, rather than combining them into a large "multi-temporal" model. Relative value measurements and management of landmark values are used to translate results from one model to another.

4.2.5 Other Types of Aggregation

Temporal abstraction is actually only one of four types of abstractions identified by Iwasaki (1992). The other three dimensions of abstraction are described as follows:

- Structural: Abstraction by lumping together a group of components that are physically close.
- Functional: Abstraction by lumping together a group of components that collectively achieve a distinct, higher-level function.
- Quantitative: Abstraction by ignoring small differences in variable values.

Iwasaki goes on to note that these "dimensions of abstractions" are not necessarily independent. For example, structural abstraction may require temporal abstraction if the aggregated structure state transitions are longer than the temporal grain size.

Both structural (on the basis of physical proximity) and functional (on the basis of role or mission) aggregation as defined by Iwasaki involve aggregating entities. Interestingly, this reference is one of the few examples of entity or process aggregation in the qualitative simulation literature.[2]

Another model dimension that can be aggregated is the functions within the model. A function defines the relationship between state variables in the model. Karp and Freidland (1989) note that just as there is a range of precision with which we can represent the values of a given variable, there is a range of precision in which the relationship between variables can be represented. They describe functions using frames which have a number of slots that contain specific information about the function. The more information provided in the frame, the more precise the function. Note the difference between this description of function aggregation and that provided by Iwasaki (1992). Karp and Freidland focus on the function without necessarily impacting the entities. Iwasaki's description of functional aggregation is actually an aggregation of entities, grouped by function. The functions performed by the entities are modified as

[2] Whether this is because such techniques have been more thoroughly researched in other simulation domains, or because they have less applicability to qualitative reasoning, is unclear to us.

required, to accommodate the higher level entities; however, it is the entities themselves that are aggregated.

4.2.6 Aggregation of Repeated Cycles of Behavior

The final behavior abstraction technique discussed in this section, introduced by Weld (1986), involves aggregation of behaviors when a repeating cycle of behaviors is detected. The fundamental principle on which this type of aggregation is based is that in some circumstances a particular cycle of behaviors in a causal simulation is repeated until some broader conditions are met. Rather then repeat the cycle, valid simulation results can be obtained with less computation by replacing the cycle with a continuous abstract process.

Weld's aggregation is a technique for recognizing when processes repeat and for generating a more abstract process description of the change over time. The inputs to aggregation include a history of parameter value changes over time, a set of active processes, the process definition, and a list of totally ordered parameters. The output is a continuous process representation of the system behavior. Cycle recognition occurs in three steps:

1. Analysis of active processes to detect situations in which the same behavior is occurring.
2. Determination of what sequences of processes are repeating; and
3. Determination of whether the cycle is truly repeating.

Note that repetition can be temporal, spatial, or both. Temporal repetition refers to behavior in which one object repeats the same action at different times, e.g., starting at times t_i, t_{i+1}, t_{i+2}, and so on. The behavior only differs in execution time. In spatial repetition, several different objects perform the same action at the same time; the behavior only differs in the object performing the behavior.

The first step is recognizing repetition. Requiring that system states be identical in comparing two process instances is too restrictive. Instead, Weld defines two process instances to be the same (for this purpose) relative to a set of predicates if all unordered parameters have identical values, and the predicates do not distinguish between the values of any of the totally ordered parameters. This distinction is necessary to allow transitional analysis to be used to generate the abstract representation of the repeated processes (Weld (1986) contains a detailed explanation). The second step, extraction of candidate cycles, first looks to see whether the objects instantiating the repeating processes are acting on different objects. If so, the cycle is spatial repetition, which is not addressed by Weld's approach. Otherwise, the cycle extractor backtracks through the state history to the previous repetition point as the candidate cycle. The final step is then to check the internal states to ensure that the cycle is truly repetitive. Weld cites some specific difficulties in this approach that require additional consideration, such as when a repeating process is acting on multiple objects, and when interdependent cycles exist.

The continuous abstract process description generated in place of the repeated cycles is based on transition analysis. Transition analysis takes as input a description of a parameter's gradual change over time and a set of boundary values for the parameter. The boundary values are possible values of parameters that are of particular interest to the modeler. Boundary values divide a totally ordered parameter space into intervals. Transition analysis determines if the parameter will move from one interval to another, and if more than one parameter is changing, which boundary value will be reached first. Using transition analysis, a new process is defined with a set of preconditions for the process and the changes affected by the process.

Weld's approach is more powerful than simple cycle recognition associated with qualitative simulation, which only checks to determine whether the global state reached by a qualitative state transition is identical to a previously generated state. Weld's definition of "sameness" is more encompassing than complete equality, and his approach also considers local states rather than the global state.

The net effect of this aggregation approach is that a set of processes that occur over multiple time intervals can be abstractly represented as a single process over a single time interval. This effect can also be considered temporal abstraction, as described in Section 4.2.4.2, since it results in an aggregation of time points and intervals (Weld and Addanki 1992).

4.3 Quantitative Extensions to Qualitative Simulation

The abstractions described in Section 4.2 involve some form of mapping multiple states into a single abstract state. The abstractions improve the tractability of the qualitative simulation by eliminating distinctions within the model that are irrelevant to the simulation requirement. In this section, we look at the definition of system state from a somewhat different perspective. We change the primary focus from qualitative states to semi-quantitative states. In this section we consider the addition of quantitative capabilities to qualitative simulation, the concept of system state in semi-quantitative simulation, and model abstractions based on that concept of system state.

The goal of semi-quantitative simulation can be pursued from two different directions. One can relax the precision of a quantitative model, moving away from strict single-valued parameters in the simulation. Parameter values, for example, can be bounding intervals, fuzzy values, or probability distribution functions. The alternative approach is to expand the role of landmark values. In a purely qualitative simulation, the landmark values do not necessarily equate to known values. Some measure of quantitative information can be added to a landmark value by mapping it to a specific value or bounding interval.

We begin this discussion by reviewing the different approaches to simulation and reasoning systems that blend aspects of both qualitative and quantitative approaches. In Section 4.3.1, we review several approaches that enhance the expressive capability by adding some quantitative information. The second section describes some approaches to working the other way, by using qualitative techniques to derive explanations for behavior simulated quantitatively. The next

subsection (Section 4.3.3) describes an approach based on applying fuzzy set concepts to a quantitative simulation to achieve a semi-quantitative capability.

4.3.1 Interval Arithmetic and Constraint Propagation in Q3

One approach described by Berleant and Kuipers (1992) involves adding some quantitative capabilities into a qualitative model. An existing qualitative structure (Q3) based on system states is maintained, and additional states and parameters are added to capture numeric values at specific events and/or times. They observe that in general, the more states that are added, the closer the model predicts system behavior. While not explicitly stated, this approach involves adding detail to a model by state addition (not decomposition). Other researchers, including D'Ambrosio (1986) and Widman (1989), have also addressed the issue of adding precision to qualitative models by defining additional states using semi-quantitative representations.

D'Ambrosio suggests adding parameters to a QP Theory model to resolve ambiguities caused by conflicting influences on a parameter. He compares this approach to the addition of landmark values in qualitative simulation. Rather than setting a landmark value as a fixed point in the variable quantity space, he adds a new parameter to the model subject to the influences similar to those of the original quantity. Like the addition of landmark values, the addition of parameters to the model provides more information in the form of a larger state space. D'Ambrosio also introduces an extension to QP Theory known as sensitivity annotations, which reflect the relative strength of an influence. This allows conflicting influences to be disambiguated based on the annotation. These annotations add information to the model, although they do not change the quantity space of the variables.

Widman more directly discusses the addition of quantitative information to a qualitative simulation for the specific purpose of enhancing the accuracy of the simulation. Specifically, he notes that "more accurate simulation models can be built by characterizing, as a second step, the numerical parameters of the [qualitative] model with precision to yield accurate simulation results." Widman also specifically cites simulation using models at varying levels of abstraction as an application of semi-quantitative models. Here again, the addition of quantitative information that improves the accuracy of the model can be considered as additional states in the state space of the model.

The common thread of these approaches is the addition of states in the model that provide some quantitative information on simulated system behavior. Additional states can be derived by decomposing states or by adding states. Decomposing states maintain the property that each decomposed state represents a condition that is a subset of the condition represented by the parent state. Additional states, by contrast, do not necessarily map back to the original states. In either case, the addition of states creates a more detailed, more quantitative model.

4.3.2 Qualitative Analysis of Quantitative Simulation

Forbus and Falkenhainer (1992) introduce an alternative approach which addresses the marriage of numeric and qualitative simulation, although from the opposite direction than the approaches described in Section 4.3.1. Rather than augmenting qualitative simulation with numeric accuracy, Forbus and Falkenhainer look to using qualitative simulation to provide explanations for results generated by numeric simulations. Their system (SIMGEN) provides the mechanism to link qualitative and quantitative models of the same process. SIMGEN takes as input a qualitative model and generates a total envisionment. This envisionment defines the qualitatively distinct regions of behavior of the system and the causal linkages among parameters.

Like Berleant and Kuipers, the key to integrating the two types of simulations is in the definition of states, with a means for ensuring (or at least checking) the consistency of the numeric and qualitative descriptions at various states. Forbus and Falkenhainer define the concept of a hybrid qualitative-quantitative state as follows. Start with a traditional numeric state definition of a set of values for continuous processes N_f. For each non-ordering proposition class in the set of propositions that define the qualitative states, add to N_f a Boolean variable whose value expresses whether the corresponding statement is true. A state is then defined as $<N_f, Q_f>$, where Q_f ranges over the set of states in the envisionment. A state is consistent if and only if the values of N_f satisfy the propositions of the qualitative state corresponding to the value of Q_f.

The self-explanatory simulation has three major components:

1. A set of procedures which specify how the numerical parameters should be updated over time for each qualitative state;
2. A set of state transition procedures which detect qualitative changes in state; and
3. An explanation facility which uses information from the envisionment to provide causal accounts and characterize possible behaviors.

Weld and Addanki (1992) also note the utility of simplification by relaxing the representation of real-valued parameters. They propose using qualitative representations instead, citing Murthy's (1988) observation that there is a natural flow from the real numbers to interval arithmetic to sign algebra.

4.3.3 Fuzzy Sets Applied to Quantitative Simulation

Shen and Leitch (1992) report similar work in the area of fusing qualitative and numeric simulations. Their motivation is to combine two approaches to dealing with incomplete information while modeling system behavior: qualitative simulation and fuzzy sets. The key to their approach is the definition of the variable quantity space. They define fuzzy quantity space, Q_f, which is generated by a finite discretization of the underlying range of each system variable. The mapping of the range of system variable values to the discrete qualitative values is

based on fuzzy numbers. The significant advantage of this approach over other qualitative simulation approaches is that purely qualitative functional dependencies are limited to monotonically increasing or decreasing functions, which are relatively weak for expressing information about the relationship. The use of a fuzzy quantitative space allows qualitative function constraints to be represented as fuzzy relations (Dubois and Prade 1980). This approach allows partial numeric information to be utilized within the functional dependencies.

The specific approach to the simulation involves generating all possible successor states, then checking the validity of the state transitions. Constraint filtering checks for consistency with the definition of the constraints and consistency between constraints that share an argument. Temporal filtering checks the rates of change of the system variables. The approach also includes global filtering to check for successor states that are identical to the previous state, and to check for repeating states.

4.4 Model Boundary Abstractions

The preceding two sections included discussions of model abstraction techniques that are applied within a model to reduce the complexity of the generated behaviors. Under such abstractions the boundary of the model is not changed in terms of what factors are considered in the model does not change with any "internal" abstractions. In this section, we review abstraction techniques that change the boundary of the model. We define such abstractions as those which modify the exogenous variables of a model.

In this section, we describe three different approaches to abstracting models by changing exogenous variables. The first approach is characterized by a pre-defined hierarchy of models, in which simpler models are abstracted versions of more complex models in which some exogenous variable(s) have been removed for simplification. In most cases these models are of physical systems, and the abstractions reflect typical simplifications in analyzing the physics of the situation (such as ignoring the effects of friction). The second approach explicitly addresses the question of where the appropriate boundary should be for the model. Specifically, we look at approaches to determine, given a library of models of components, which model components should be incorporated into an overall model of the system, based on the influences of processes on variables. The third approach is based on a recognition that valid results may be obtained from a simplified model that only approximates the results of the more complex model, if the difference is bounded and within the tolerance required for answering a specific query about the system. Approximation approaches are described in Section 4.4.3.

Semi-Quantitative System States and Numeric Representation

There are a number of approaches that are characterized by a quantity space with some mixture of intervals (both strictly defined and "fuzzily" defined) and specific numeric values. Note that this definition holds regardless of whether the approach is based on adding quantitative specificity to a qualitative model or providing generality to a quantitative model. This in turn suggests that there is a continuum of state representations, ranging from purely qualitative to purely quantitative, which provides a dimension for model abstraction.

In general, the more quantitative the model is along this continuum, the larger the quantity space (i.e., the more possible values that a variable can assume). The larger quantity space increases the number of possible system states. Conversely, the more qualitative the model, the fewer the number of possible system states. Thus models can be abstracted by reducing the number of possible system states by moving in the qualitative direction along this continuum. There are a number of approaches to reducing the number of system states; each approach serves as an abstraction technique that moves the model in the qualitative direction.

The concept of quantity space and a qualitative/quantitative continuum leads to an observation concerning numeric representation in model implementation. Consider the difference between a model implemented using integer values and the same model using double precision floating point representation. As precision of the representation increases (from integer to single precision to double precision), the domain of possible states increases, and as precision is decreased, the state possibilities are reduced. A discretization of the quantity space of real numbers into a space of integer values can be expressed as follows (x is a real number, I is an integer):

$$x \in I \quad \text{if } I - 0.5 \leq x < I + 0.5$$

Note that computer arithmetic is still limited to a finite discretization of the set of real numbers based on the number of bits used to represent a value. In this sense, all discrete computer simulations are qualitative (or at least have a finite number of system states which facilitates expressing the ordinal relationships of variables). This suggests that many of the concepts for behavioral abstraction should be applicable to discrete event type simulations, even if the mechanisms described in the qualitative simulation literature cannot be directly applied.

4.4.1 Model Hierarchies with Explicit Assumptions

The fundamental notion of this category of model abstraction techniques is based on a pre-defined hierarchy of models or model components, in which the relationships between models in the hierarchy are defined as explicit assumptions. For example, a pre-defined hierarchy of models could include one simplified frictionless model as well as a more complex model in which friction is explicitly addressed. The assumptions in this case tend to drive the models; if the abstraction of friction is the explicit assumption, an appropriate model that represents the more simplistic view of the world must then be adopted.

4.4.1.1 Graph of Models

The Graph of Models concept of Addanki, Cremonini, and Penberthy (1991) uses a graph structure to organize these explicit assumptions and associated models. The structure then provides a framework for automatically traversing the model space in search of a model which best fits the simulation requirements for the application of the model. In the Graph of Models concept, the nodes of the graph represent models, and the directed edges between nodes represent abstractions. Abstractions are explicit assumptions about factors to be included or excluded from consideration.

As an example, Addanki et. al. (1991) use the temple door mechanisms of Ancient Greece, which requires kinematic and thermodynamic models to represent the system behavior. A hierarchy of models with progressively complex system representations includes the following:

- a simple model that assumes no friction and uniform mass distribution (i.e., the model abstracts these factors from consideration);
- a more complex model that includes the effects of friction; and
- an even more complex model that includes the effects of friction and torsional bending.

Abstractions (or in their terminology, assumption transitions) are defined as parameter change rules, which indicate the overall effect that changing an assumption has on a parameter. For example, a model that accounts for rolling friction decreases the net torque transmitted from one gear to another relative to a model that ignores friction.

The Graph of Models concept defines the relationships of a set of models in terms of these parameter change rules. When a model is used to predict behavior of a physical system, model predicted behaviors are compared to observed behaviors. Differences in observed behavior and predicted behavior are then used to navigate through the Graph of Models to select a model that better predicts system behavior. Differences in predicted and observed behavior, called conflicts, are characterized in terms of the parameter and the direction in which the value of the parameter must move to eliminate the conflict. These delta-vectors are then analyzed with respect to parameter change rules to determine which assumptions best "bring" the predicted parameter values closer to the observed values (for observed values outside a suitable range). The analysis

thus determines which edges to traverse in the Graph of Models to arrive at a model with fewer (or no) conflicts.

4.4.1.2 Compositional Modeling

Falkenhainer and Forbus (1991) have developed an approach which includes many of the characteristics of the Graph of Models concept. The most important common characteristics, in terms of this taxonomy, are pre-defined model fragments and explicit assumptions concerning the factors that are represented in a model.

In this approach, a model fragment is defined as a quadruple $\langle I, A, O, R \rangle$, where:

- I is a set of conditions defining the structural configuration of individuals to which the model applies.
- A is a (possibly empty) set of assumptions (such as frictionless flow).
- O is a (possibly empty) set of operating conditions delimiting the model's behavioral scope.
- R is the set of relations imposed by the model fragment, which can include qualitative physics (QP) theory constraints and numerical constraints.

Model composition consists of assembling a model from model fragments in response to a user query. The model is constructed to be sufficient to answer the query with minimal effort. Sufficiency requires that the model includes all relevant aspects of the system, and that there is enough detail to provide a suitable answer. Model fragments are selected based on the following approach. First, model fragments that are applicable to the query are identified. Then additional objects required to express all relevant interactions are identified. Choices of simplifying assumptions are identified, completing the identification of candidates. The candidates are then evaluated, and the "best" candidate selected.

There are several unique aspects of compositional modeling, including the following:

- Model fragments, rather than "complete" models, are the basic building blocks, allowing models to be composed.
- Composition facilitates modeling of different aspects of the system at different levels of fidelity.
- Model fragment selection and composition are performed in response to user specifications, in addition to comparison of predicted and observed behaviors.
- There are specific assumption classes.

In Compositional Modeling, it is the definition of types of assumptions that is the key to model abstractions. Assumptions are categorized as either simplifying assumptions or operating assumptions. Operating assumptions specify conditions, such as valid ranges of input values,

under which the model holds. Simplifying assumptions, as their name suggests, provide potential for abstraction techniques. Four types of simplifying assumptions are defined:

- grain assumptions: controls amount of the physical system about which to reason;
- perspective (or ontology) assumptions: controls point of view;
- approximations: defines simpler model with reduced accuracy; and
- abstractions: reduce model complexity without reducing accuracy, but with less detail[3].

Grain assumptions are the basis of aggregating objects into composite objects where detail is not required. Objects are the only dimension of aggregation identified in Compositional Modeling. This notion of grain assumptions actually fits the concepts described in Section 4.2.1 on Behavior Aggregation.

Ontological assumptions define the "point of view" adopted by the model. Ontological assumptions dictate fundamental aspects of the model, such as coordinate system, analysis of data over time and/or space, analysis primitives, and so on. For example, in a thermodynamic model, different ontologies can include a contained stuff ontology (an Eulerian view of fluids and their containers), an energy flow ontology, a molecular collection ontology (a Lagrangian view following the movement of a localized unit during flow), or a mechanics ontology. Ontologies are fundamental, and thus must apply to a composed model, not an individual model fragment. They also represent different problem perspectives, rather than more or less complex implementations of the same problem perspective. Thus ontological assumptions do not fit well with our notion of abstraction. However, within a Graph of Models structure, ontological transitions could be maintained as a means to relate different problem perspectives (even without a sense of vertical movement through a hierarchy of more or less complex models).

Ontological perspective is incorporated into the model composition process under a concept described by Nayak, Joskowicz, and Addanki (1992). They propose the use of context-dependent behaviors (CDBs) as a behavior modeling representation for encapsulating contextual modeling constraints. These constraints express both structural and behavioral constraints, and can be organized in a generalization hierarchy, a possible models hierarchy, and assumption classes. Structural constraints express structural preconditions for the model, such as physical properties and interfaces. The interface specification ensures that only compatible CDBs are incorporated into a device model. Behavioral preconditions in a CDB are constraints on the behavioral context of a component that must be satisfied if the component is to be modeled by the CDB. In addition, behavioral constraints can enforce selection of additional or related CDBs.

[3]This definition of "abstraction" encompasses a narrower domain that our use of the term abstraction (as in the title of this contract). The remainder of this section follows the terminology of the paper being summarized, and uses the term abstraction as defined here.

Expected behavior, describing the function of a component, can also be specified; this ensures that related CDBs that are required to fulfill the function of each component are included.

The ontological perspective is achieved by defining constraints in a possible models hierarchy. Figure 3, taken from (Nayak, Joskowicz, and Addanki 1992) shows a possible models hierarchy for a wire. The different models are based on different structural and behavioral constraints levied on a model of the same physical component. Within this hierarchy organized by ontology, some of the models fit the concept of explicit assumptions. For example, the perfect-conductance model of a wire is (or can be) an abstraction of the "resistor" model, in which an explicit assumption is made to ignore the effects of resistance within the wire. Similarly, "constant-resistor" can be considered an abstraction of "temp-dep-resistor" in which the effect of temperature is abstracted. Also note that there is a many-to-many relationship

Reuse Features of Compositional Modeling

The assumption classes are the feature of compositional modeling that support the application of model abstraction, but the other features are worth noting because of their contribution to model management and model reuse, as described by Frantz and Brown (1992).

Model fragments. The notion of using model components is of interest from the broader context of model management as well as the specific abstraction issues addressed by this research effort. Model composition requires a definition of the manner in which the models interact. This is accomplished by the definition of shared individuals and parameters. In addition, a general class of partial constraints, called composable functions, is used to partially specify relationships among parameters that cannot be fully specified within a model fragment without the additional contextual information of the aggregate model. Compositional modeling specifically uses two composable functions introduced by QP Theory (Forbus (1984), called influences.

Modeling at different levels of fidelity. By defining each model fragment in terms of individuals, it is possible to define multiple fragments with the same parameters and interfaces, but based on different assumptions that represent different levels of fidelity. Each component of the physical system is mapped to an individually instantiated model fragment; thus each physical component can be modeled as required to satisfy a user's query. Flow in one pipe in a model could be modeled assuming a viscosity factor, while flow in another pipe could be modeled ignoring such a consideration. This approach also provides a means to aggregate some objects if their detailed behavior is irrelevant to the query at hand, using granularity assumptions.

User specification. Compositional Modeling provides a means for defining the requirements of the model, expressed as a query posed by the user. The queries described by Falkenhainer and Forbus (1991) are specific (as opposed to more general statements of simulation requirements). For example, for a multi-grained, multi-perspective model of a shipboard steam-powered propulsion plant, the queries can be posed to the system, such as:

- What affects the efficiency of the plant?
- How many mass flows are there?
- What is causing black smoke to rise from the furnace?

Response to each of these queries involves selection of assumptions that ignore insignificant detail and identifies the model fragments necessary to provide the response.

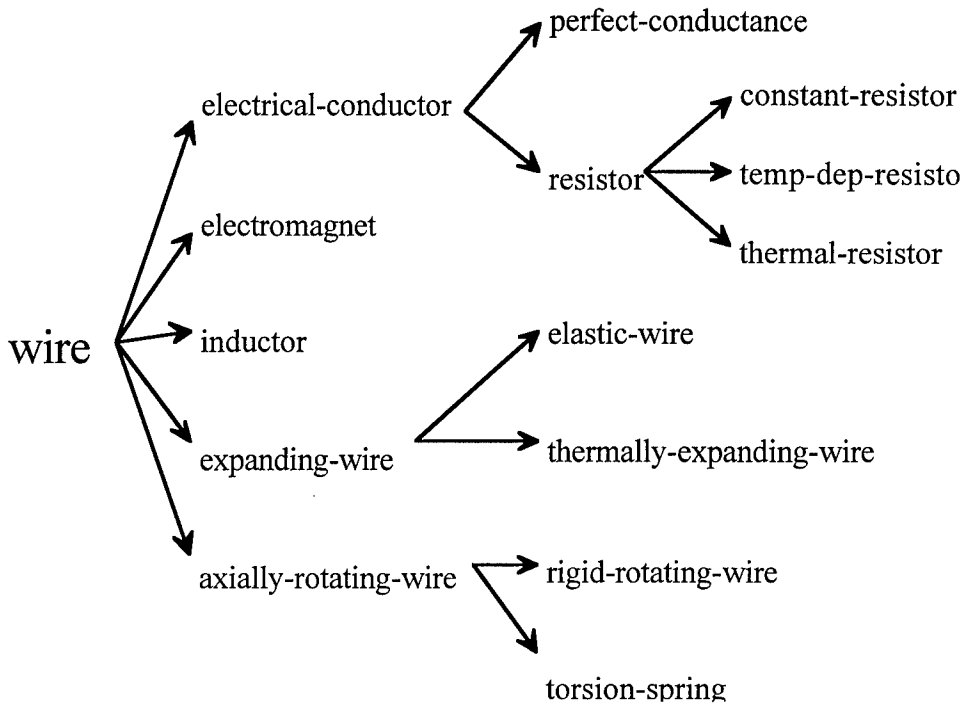


Figure 3, Example Organization of Models

between physical components and CDBs. Clearly a single component can be modeled by many CDBs; a CDB describing electrical conductance can apply to several physical components.

4.4.1.3 Other Model Hierarchy Concepts

Other authors have also described model abstractions in a hierarchical form. For example, Agrawal (1985) describes a hierarchical structure for a set of models of queuing networks. He addresses the concept of defining models of reduced computational complexity which approximate the model outcomes of the more detailed model or actual system to within an acceptable tolerance. The specific domain under consideration by Agrawal is that of analytic queuing network models. Most of his examples consist of defining the transformations applied to mathematical models to implement the impact of a specific assumption. He points out that the transformations applied to a model can be applied again to the initial transformation product to create a hierarchy of models. The hierarchical structure and explicit assumptions are similar in concept to the Graph of Models approach (see Section 4.4.1.1), although Agrawal concentrates on the mathematics of the assumptions, rather than methods for selecting a minimally acceptable set of assumptions.

4.4.2 System Boundary Selection

The Graph of Models approach provides a means to reason about which of a hierarchy of existing models provides the best model for reasoning about system behavior. Compositional modeling provides a way of using existing model fragments and building a minimal model to meet

the requirements. In this section, we review an approach based on compositional modeling to automatically select a time scale and system boundary for a model.

4.4.2.1 System Boundary Selection Concepts

Compositional modeling lends itself to an analytic approach to determining the elements of a model that are required to answer a particular question, or meet a particular simulation requirement. Rickel and Porter (1994) frame this question as determining the system boundary; that is, determining which variables require representation in a model, and which of these variables can be considered exogenous to the system. They characterize the problem as choosing "a system boundary that separates aspects of the scenario that must be modeled from those that can be ignored." The objective in system boundary definition is to identify the simplest model that can be used to accurately answer prediction questions. In our terminology, this same objective can be stated as identifying the most abstract model that meets simulation requirements.

The system boundary selection approach described by Rickel and Porter (1994) is based on searching the space of all possible models. The models in the search space are distinguished by the assignment of all model variables into one of three categories: exogenous, dependent, or not of interest. Variables in the model are related by influences (causal relationships between variables). The initial state in the search is a model in which all variables of interest are considered free, meaning that they have not been chosen as exogenous or dependent variables. Candidate models are postulated by choosing a variable that must be dependent and determining the combinations of candidate influences on that variable. Note that such combinations may include variables not initially considered of interest. Candidate models that violate coherence constraints are eliminated from the search space. The search continues until all remaining free variables can be considered exogenous (while meeting adequacy criteria), at which point the simplest model has been determined. (Simplest in this case is defined as having the fewest variables; other simplicity criteria could be used, changing the ordering of the candidate models in the search space but not changing the overall approach.) The details of the model selection algorithm, such as how coherence criteria are checked and how the model space is searched, are described by Rickel and Porter (1994).

4.4.2.2 Time Scale Selection Concepts

The time scale of an influence is a significant factor in determining system boundary. Candidate influences which occur over a slower time scale than the time scale of interest are considered insignificant and are not considered in the determination of whether a variable is dependent. Thus influences that occur over a time frame that is too slow to impact the specific question being addressed are thus abstracted out of the model to be used to answer the question. Rickel and Porter include an algorithm for automatically selecting a time scale for answering a specific question based from the candidate time scales defined for the various influences in the model. A time scale is considered adequate if, at that time scale, every variable of interest is significantly influenced by some variable referenced in the question's behavioral conditions.

The implication of time scale on selection of system boundary can propagate other abstractions within a model. For example, processes which change at a time scale faster than the time scale of interest can be modeled as static approximations of the equilibrium conditions of the processes. Also, entities whose behavioral distinctions are a function of "too fast" processes can be aggregated.

4.4.3 Approximation

An alternative approach to eliminating factors from consideration within a model is to eliminate exogenous inputs to the model whose effect on the simulation does not affect the overall conclusions of the simulation. This approach assumes that one can use a simpler model to generate an approximation of the results of the more detailed model. It is also assumed that the difference between the approximate results and the results of the more detailed model are insignificant relative to the simulation requirement. Stated another way, there is a tolerance within which the simulation outputs can vary without invalidating the conclusions of the simulation; approximation eliminates exogenous parameters that vary the simulation outputs only within that tolerance.

In this section we present two different approaches to approximation. First we review an approach based on approximations of structures represented in the model. We conclude with a discussion of approximation based on a sensitivity analysis of the accuracy of a model.

4.4.3.1 Structural Approximation

Selecting the model components that comprise the simplest model necessary to answer a question is a challenging aspect of model composition. Nayak (1992) argues that finding the simplest model that meets the simulation requirements is an intractable problem, and introduces the concept of a causal approximation to structure the search in a solvable way. The causal approximation establishes some rules about how model components can be selected. The causality of a causal approximation is based on causal ordering such that parameters are determined by equations in an orderly sequence, and every parameter is expressed as a function of other parameters. The approximation is defined in terms of a simple model that is in the same assumption class as a more complex model that describes the same physical process. By organizing models in terms of their causal approximation relationships, selecting the appropriate set of model components is a polynomial time problem.

4.4.3.2 Model Sensitivity Analysis

Both the Graph of Models and Compositional Modeling approach require as a starting point a set of models or model fragments, and a definition of the assumptions on which the models or model fragments are based. These approaches provide powerful tools in automated or semi-automated selection of efficient models, but are limited in terms of the model domain space. No flexibility exists to customize the model to suit the problem at hand. An alternative approach is to analyze models to determine which parameters in a model can be eliminated, yielding an

approximation which is computationally simpler yet provides approximate results that are sufficient to meet the requirements of the simulation.

This section describes an approximation approach based on model sensitivity analysis described by Weld (1992). Weld's motivation in developing this approach is to provide a capability for model abstraction that does not require a priori definition of the model hierarchy and abstractions. Many of the qualitative simulation and reasoning approaches previously described in this section are based on a pre-defined hierarchy of models with explicit assumptions. The limitation of these approaches is that the model resolution is driven by model capabilities rather than simulation requirements. Customization of a model hierarchy can only be accomplished within the strict definition of the existing model hierarchy. Weld's motivation is to develop techniques that can be customized based on the simulation requirements.

The basis of Weld's approach is Model Sensitivity Analysis (MSA). The cornerstone of the MSA approach is the notion of predicting how a change in models will affect the resulting predicted behavior over time. Fundamentally, MSA takes three inputs, a source model (P), a destination model (Q), and a variable of interest (p_i), and generates a description of the difference between source and destination model's predictions of the variable of interest's behavior over time. MSA can then be applied to determine whether the behavior represented in Q is insensitive to the transformation from $P \rightarrow Q$.

The challenge in applying MSA is the computational burden for solving this problem in the general case, or conversely, the limited domain that can be addressed by computationally feasible approaches. For example, one can solve the dynamical constraints of P and Q for closed form solutions to the respective behaviors of interest, and use the resulting difference as the description of the variables' behavior. However, this approach is both computationally demanding and limiting in scope.

To facilitate the application of MSA, Weld introduces a class of model relations called **fitting approximations**. The problem of computing the differences between two models is then reduced to one of computing the sign of partial derivatives in a single model. With the capability of efficiently performing model sensitivity analysis, he then develops a theory of query-directed model simplification. Approximations can also be retracted, a process which is referred to as refinement, and a theory of discrepancy-driven model refinement is also developed. Refinement creates more complex, and more accurate, models.

Weld defines a fitting approximation as follows: "a simple model, P, is a fitting approximation of a more complex model, Q, when the complex model has an extra exogenous variable, called a fitting parameter, that allows Q to be tuned so that its predictions match those of P's to an arbitrary tolerance."

The concept of query-directed simplification is expressed in terms of a model, Q, and two parameters, X and Y. A query is considered an inequality between the values of two parameters: $X < Y$. Consider a simpler model, P, that overestimates X and underestimates Y; if the inequality

is true in P, then in must be true in Q. In other words, if we have a model Q designed to affirm that $X > Y$ (e.g., that wing strength exceeds worst case load) and we have a simpler model P, then we can safely use P in place of Q as long as we know that P consistently underestimates X and overestimates Y. Then an affirmation by P guarantees an affirmation by Q (e.g., wing strength is adequate). A "not affirmed" result of P indicates that Q must be executed to determine the answer. Using a full range of inequalities and logical relations, a powerful approach can be followed for simplifying models without altering the predictions that the models make.

Discrepancy-driven refinement is the dual to the simplification process. The refinement strategy starts with a simple model and retracts assumptions until the model's conclusions are consistent and agree with the data. This process is applied in situations where discrepancies are identified between predictions made by a model and actual data collected from observations of the system being modeled. The discrepancies represent the numerical implication of the abstraction process; some real world factors have been ignored in the model, resulting in the discrepancy. By identifying and analyzing the discrepancies, it is possible to determine which assumptions made in the abstraction process led to the discrepancy, and should be retracted. Retraction of an assumption is logically equivalent to adding another factor into the model, i.e., making the model more complex.

4.5 Comparison of Traditional Discrete Event and Qualitative Simulation

The preceding sections describe a variety of approaches to performing abstractions of qualitative simulation models. The qualitative simulation domain typically involves continuous real-world processes that are conceptualized as ordinary differential equations, and expressed as qualitative differential equations. As indicated previously, our interest is in how these ideas of model abstraction could be applied to C^3I models. Part of the research into qualitative reasoning about physical systems has addressed using multiple models of such systems that express parametric relationships at varying levels of detail and complexity. In many ways this problem corresponds to our problem of dealing with models with multiple levels of fidelity. Consider, for example, this expression of the problem by Addanki, Cremonini, and Penberthy (1991): "Each model is a reformulation of domain knowledge that can only be used if its assumptions lead to an acceptable approximation of the world. Our goal in using multiple models is to simplify problem solving by permitting analysis in the simplest model that is an acceptable approximation of the world." In this section, we consider the similarities and differences between qualitative and quantitative discrete event simulation. This comparison sets the stage for the combination of abstraction techniques from both disciplines in Section 5.

We first consider the common ground between the two domains. There are clearly analogs between qualitative simulation and discrete event simulation that we can exploit. First, the description of a physical system in terms of qualitative constraints is identified as an abstraction, much as we define a conceptual model of a physical system as an abstraction. Kuiper's (1994) definition of system is narrower than our (rather general) definition, in that Kuipers specifically

defines a system in terms of "reasonable" functions that behave well enough to apply the concepts of qualitative simulation. Kuipers also defines a notion of system state, which consists of a qualitative value (the relationship of the state variable to landmark values) and the qualitative rate of change of the variable. The analogous definition of state in the discrete event simulation domain is the numeric value of the variable.

There are also significant differences between traditional (including both discrete event and continuous) simulation and qualitative simulation, as follows:

1. The representation of state variables;
2. Continuous versus discrete state variable changes;
3. Finite versus infinite system states; and
4. The underlying motivation for the models.

The significance of each of these differences is discussed in a paragraph below.

The most significant difference is that fact that the models in qualitative simulation derive predictions of behavior from non-quantitative models The states of a qualitative model are not defined in terms of variables having specific, quantitative values. Thus qualitative models represent classes of systems rather than complete system specifications typical of numerical simulations (Berleant and Kuipers 1992). In most cases exact information about the physical system is not available for the reasoning system, so prediction of the next state of the physical system is based on characteristic relationships of parameters of the system, rather than a numerical calculation.

Another significant difference is that the processes represented in C^3I simulations are not necessarily continuous, and there are a variety of interactions of entities that cannot be expressed as differential equations. Thus many of the mathematical foundations for qualitative simulation can not be directly used in the discrete event simulation domain. Formal proof, particularly in the area of preservation of the validity of the simulation outputs, could be developed by structuring analogous proofs in the framework of the existing discrete event simulation formalisms.

A third significant difference is that qualitative models generally have a finite number of system states, and often the simulation consists of generating all possible states (the total envisionment) or at least the set of all states that can be reached from the initial conditions (the attainable envisionment). In contrast, discrete event models generally have an infinite number of system states, and the simulation consists of generating one of the possible state trajectories (for each simulation run). Abstraction techniques that operate on the complete envisionment of a qualitative model cannot be directly extended to the discrete event simulation domain, at least in an automated way. The best that can be accomplished is to develop a "manual" process (i.e., performed by an analyst) that looks for the same abstraction opportunities that can be found automatically in the envisionment of a qualitative model.

The final difference that we identify is more philosophical, but reflects a divergence in the motivations of researchers in the communities. One of the perspectives that Zeigler (1990) advocates is that simulation models can be treated as knowledge. Specifically, models encode knowledge of the effect of intervention on a system. Their sine qua non is the ability to integrate sets of disconnected relationships whose joint implications would otherwise be difficult to draw. The value of models is in their ability to represent relationships that cannot be easily reasoned about by the human mind. This philosophy presents an interesting comparison with that of the AI qualitative reasoning community. Their models represent relationships that can be easily reasoned about by humans, but for which the reasoning is to be automated. Thus the qualitative reasoning models tend to deal with much simpler physical situations, but with a greater emphasis on automatic creation and manipulation of the models themselves.

Having emphasized the differences between discrete event and qualitative simulation, we close this section with an observation about their commonalty. The fact that in both cases the models are ways of capturing knowledge about a system provides a unifying theme. Rather than considering these approaches to simulation as diametrically opposed or divergent, we consider them complementary approaches to expressing and utilizing system structural and behavioral knowledge.

5. A Taxonomy of Model Abstraction Approaches

This purpose of this section is to organize the various model abstraction techniques identified and summarized in previous sections into a taxonomy. There are a number of reasons why a taxonomy is appropriate and useful for this research. Our motivation in developing this taxonomy is threefold. First, a taxonomy helps provide a common framework in which various techniques can be compared. These techniques are drawn from different fields of study, and a coherent organization and definition of distinguishing characteristics is necessary to develop synergy among these disparate techniques. Second, it is the first step in the process of characterizing the circumstances in which various techniques should be applied. Third, it provides the basis for insight into how techniques might be combined.

We are not introducing any new techniques in this section; all of the techniques have been introduced by one or more researchers in existing literature. Some elements of the taxonomy are broadly drawn from existing categorizations, although no comprehensive taxonomy exists. Much of this study is based on the premise that, although some techniques have been developed for continuous, deterministic simulation models, they can be extended and adapted for use in the simulation domain of interest to Rome Laboratory (discrete event, stochastic models).

We developed this taxonomy on the basis of the techniques used to simplify a model. In general, the grouping of techniques is based on how the structure of the model changes to create the simplification.

The taxonomy is shown in Figure 4. The first level reflects the distinction between abstraction of behaviors within a model and abstraction of the model boundary. **Model boundary abstraction** involves:

1. Changing the model boundary; or
2. Changing the exogenous variables to the system; or
3. Changing the real-world factors accounted for by the model.

Each of these thoughts expresses essentially the same concept.

From the domain of C^3I models, consider the example of an air-to-air combat model. An abstract version of a detailed model might abstract (ignore) the position of the sun relative to the aircraft pilots. This is a change to the model boundary, since the abstracted model no longer includes the processes or entities associated with the position of the sun. Sun location, or at least time of day and year need not be input to the model, changing the set of exogenous variables of the model.

Model behavior abstraction eliminates the transitions through system states whose distinction is irrelevant to the simulation requirement. For example, a model of the movement of military units might be represented at the detailed level by calculating the location of each individual vehicle at (frequent) periodic intervals. However, if the only information relevant to

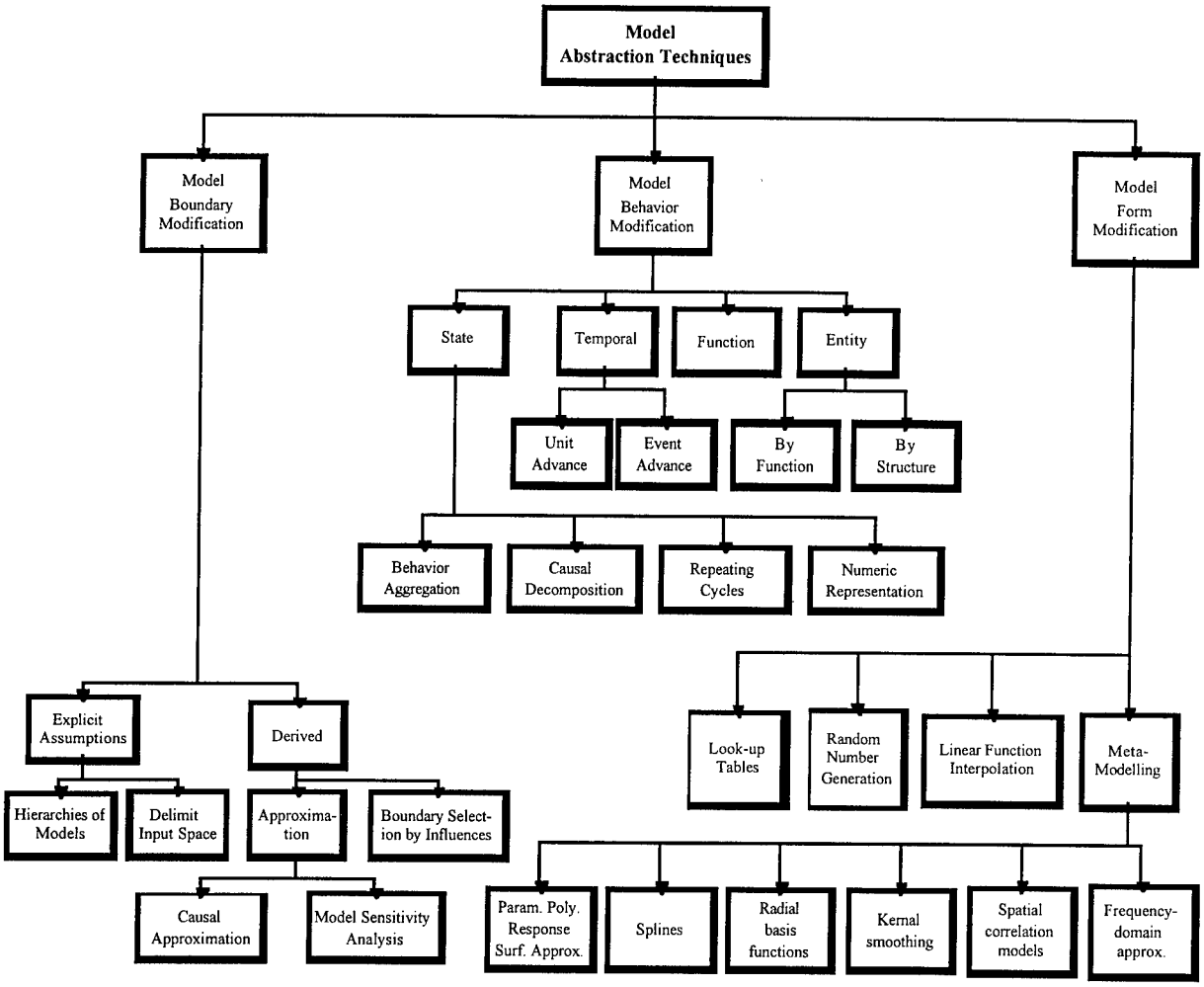


Figure 4, Taxonomy of Model Abstraction Techniques

the simulation requirement is the arrival time at a particular location, the model can simply calculate the transit time. This approach results in aggregation of a number of detailed system states (which describe the units' intermediate locations) because their distinction (the different locations of vehicles over time) is irrelevant to the simulation requirement. Note that the simulation requirement is the controlling factor here. If for the purposes of the simulation it is important to reflect situations that could occur during unit transit, then the intermediate states cannot be ignored. In this example the exogenous variables, the factors that influence the length of time required for the unit to move, are not affected by the abstraction. The abstraction could not be made if an exogenous variable represented the possibility of a road being subject to opposing force air interdiction strikes. For this reason, such an abstraction is referred to as a behavior abstraction rather than a model boundary abstraction.

In addition to behavior abstraction and model boundary, we identify a third category of abstractions at the top level that do not fit into either category. In some cases it is possible to

abstract a model without changing either the system boundary or the behaviors, but to determine input-output relationships in a different way. This category is identified as **model form abstraction.**

The discussion in this section is organized in three subsections, each focused on one of these three major abstraction classes identified in the preceding paragraphs. We then conclude this section with a discussion of the relationship of this taxonomy to other categorizations and organizations of abstraction techniques.

5.1 Model Boundary Modification

The first class of model abstraction techniques is based on modification of the input variable space, typically the elimination of variable parameters input to the model. At the conceptual model level, this category of model simplification amounts to removing a factor from consideration. For example, a particular model of a physical system could be simplified by ignoring the effects of friction. Within the category of parameter elimination techniques, we define two subcategories: explicit abstractions and derived abstractions. Explicit abstractions require the modeler to specifically identify the exogenous variables to be eliminated in the abstraction. Derived abstractions, on the other hand, are determined by analyzing a particular model or a particular set of input variables, and determining the minimum set of those variables that are required to meet the simulation requirement. Each of these categories of abstraction techniques is described in a subsection that follows.

5.1.1 Explicit Assumptions

The first category of assumptions deals with those that are expressed explicitly by the model developer. Within this category, we include two subcategories: hierarchies of models, in which the abstractions are expressed as specific relationships between models; and limitations of input ranges, in which assumptions are expressed as constraints on the ranges of input variables. Each of these approaches is discussed in a subsection below.

5.1.1.1 Hierarchies of Models

This category of model abstraction techniques is based on a pre-defined hierarchy of models or model components, in which the relationships between models or model components in a hierarchy are defined as explicit assumptions. For example, a pre-defined hierarchy of models could include one simplified frictionless model as well as a more complex model in which friction is explicitly addressed. The assumptions in this case tend to drive the models. The abstraction of friction is the explicit assumption, and an appropriate model that represents the more simplistic view of the world must then be developed or incorporated.

This particular approach is popular in the field of qualitative reasoning, perhaps because the roots of qualitative reasoning are based on defining models of physical system interactions, which are often described in terms of increasing or decreasing complexity. For example, there are

numerous "ideal" laws in physics which "abstract" away some particular complicating factor to allow a simpler expression of the interaction. Since such laws are often the basis of systems used for qualitative reasoning, it is not surprising that the notion of abstraction by parameter elimination is used. Specific examples of pre-defined models with explicit assumptions that add or remove parameters from consideration include the Graph of Models concept of Addanki, Cremonini, and Penberthy (1991) and Compositional Modeling of Falkenhainer and Forbus (1991).

5.1.1.2 Delimit Input Space

Another abstraction technique that affects the model boundary is based on limiting the domain of the inputs provided to the model. We refer to this technique as delimiting the input space. In Zeigler's (1976) terminology, this technique is equivalent to defining the experimental frame of reference within which the model is valid. A model can be abstracted by limiting its applicability to a subset of the input variables. For example, the model can be limited to a range subset, such as a model that only applies if the water temperature is in the range of 0 to 100 degrees Celsius. Computation associated with transition of water to solid or gas can be eliminated, thereby simplifying the model. An input variable range can be restricted to a single value, i.e., making it a constant rather than a variable.

5.1.2 Derived Abstractions

The second category of abstractions that modify model system boundaries is referred to as derived abstractions. They are so named because they are techniques for taking an existing model and/or input variable set and deriving whether and what simplifications can be made. They do not depend on an a priori decision made by a model developer, which has the advantage of not requiring the model developer to recognize a priori the potential abstraction opportunity. We have identified two types of derived abstraction approaches in the literature: approximations, and a technique based on the influences of input variables. Each of these classes of derived abstraction techniques is discussed in a subsection below.

5.1.2.1 Approximation

An alternative to the pre-defined model hierarchy is parameter elimination based on the requirements of the simulation to be executed, rather than defined externally. In our taxonomy, this branch is characterized by techniques that eliminate parameters based on analysis of existing models. Such an analysis must determine which parameters in a model can be eliminated, yielding a new and computationally simpler model that provides approximate results that are sufficient to meet the requirements of the simulation.

We have identified several approaches in the literature, including Causal Approximation (Nayak 1992) and the Model Sensitivity Analysis approach defined by Weld (1992). Both these approaches are described in Section 4.4.

5.1.2.2 Selection by Influences

Another technique for deriving the simplest model to address a particular simulation requirement involves selection by influences, as described in Section 4.4.2. This approach focuses on determining which variables in a model must be exogenous variables, and requires that the influences of variables on other variables in the model be defined.

5.2 Modification of Model Behaviors

The second major category of abstractions is defined as approaches that involve modification of model behavior. The defining difference between these abstractions and those described in the preceding section is whether the abstraction changes exogenous variables or internal elements of the model. In Section 5.1, we addressed modifications to the model input space that do not necessarily involve internal changes to the model, only the data that is input to the model. In this section, we address abstractions that change the structural aspects of the model without (necessarily) changing the inputs to the model.[4]

Behavior abstraction involves aggregating some aspects of the model. There are a number of model aspects that can be aggregated. We therefore create within the taxonomy another level of decomposition of abstraction techniques based on the aspect of the model that is aggregated: states, time, entities, and functions. Each of these areas is discussed in more detail in a subsection below. Note that such aggregations are not mutually exclusive, and aggregation of one model element may require concomitant aggregation of another element.

5.2.1 State Aggregation

The definition of aggregation, as opposed to approximation, is based on the definition of the states of a model. We define a model state as: an n-tuple of system state variables. Given a discrete event simulation model and a finite representation method for state parameter values, any discrete event simulation model has a very large but finite number of possible states. One way to define a simpler model is to define a model with fewer possible states.

[4]The use of the term "necessarily" here reflects the fact that abstractions can change both the input parameters and the internal structure of the model. We address this aspect of abstractions in Section 10.2.

The state of the abstract model consists of, or can be decomposed into, one or more states in the refined model. Conversely, all states in the refined model map to a state in the abstract model. This notion of aggregation, drawn from the domain of qualitative simulation, is illustrated in Figure 5. At the more abstract level, there is a single qualitative state in which the value of X is greater than the value of Y. This aggregate state is composed of five states, where the addition of 0 as a landmark value provides the basis of the decomposition. The five "lower level" states are subsets of the higher level state.

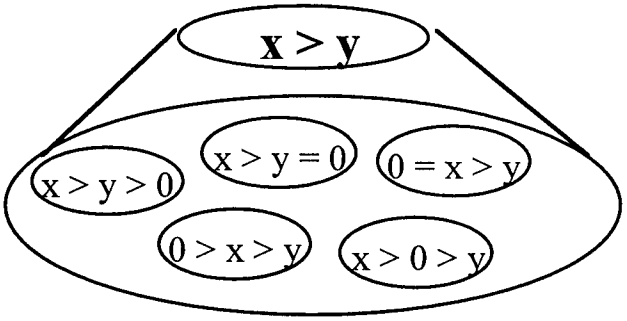


Figure 5, State Aggregation Example

Zeigler's (1976) definition of coarsening of range sets fits our concept of state aggregation. In Zeigler's terms, "the lumped model variables can describe relatively few of the conditions that the base model components can be found in." In our terminology, replacing lumped model with abstracted model, base model with original model, and conditions with state, this statement is as follows: the abstracted model has fewer states than the original model, i.e., the states have been aggregated.

Consider, for example, a model of a networked computer system. A detailed model could include various states for each processor, such as "operational" along with a number of failure conditions. If the model were executed for a particular application that was only concerned with modeling overall throughput, the states representing individual failure conditions could be aggregated into a single state, called "not operational." The example of model abstraction in Section 2.2 is also an example of state aggregation. The states which are distinguished by parameters such as the number of people in the checkout line but have the same time until the checker is no longer busy are aggregated into a single state.

There are a number of approaches to determining how to aggregate states. Four specific approaches identified in the literature include the following: behavior aggregation, causal decomposition, aggregation of cycles, and numeric representation. Each of these approaches is addressed in a subsection that follows.

5.2.1.1 Behavior Aggregation

The principle of behavior aggregation is to combine states whose distinctions are irrelevant to the simulation outcome. Consider a transition graph representation of simulation history, as shown in Figure 6. Each system state is represented as a node in the graph, with arcs representing transitions among states. A single input single output subgraph is a candidate for abstraction to a single system state, removing the necessity to compute the separate states of the subgraph. Consider the example of a military troop movement model described in the introduction to Section 5. State S2 in Figure 6 represents the initial location of the unit; states S3 and S4 represent intermediate states following different routes of march. State S5 represents the unit at the destination location and time. If the distinction between S3 and S4 is irrelevant (meaning that the simulation results are only dependent on whether the unit arrives at the destination location by a certain time), then the model can be abstracted to eliminate the decision logic to determine the route of march.

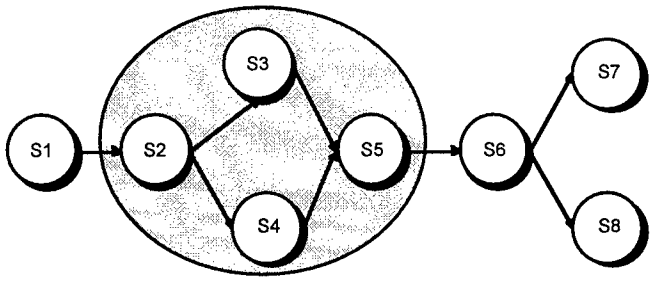


Figure 6, Notional Behavior Aggregation Example

5.2.1.2 Causal Decomposition

The philosophy of causal decomposition introduced by Clancy and Kuipers (1994) is to divide a model into loosely connected components, then execute each component separately while specifically addressing the component interactions. The causal relations among components are analyzed such that execution of one component can constrain the behavior of subsequently executed components. Also, where there is no causal relationship the components can be executed in parallel. Causal decomposition is implemented as three major steps: variable partitioning, creation of submodels, and then executing the partitioned simulation.

5.2.1.3 Aggregation of Cycles

One aggregation approach developed in the domain of qualitative simulation is aggregation of cycles (as described in Section 4.2.6). The aggregation concept is to combine the states of an envisionment that reflect similar sequences in which distinctions among the individual sequences are irrelevant to the final outcome. For example, an iterative determination of a situation (such as checking the distance between two closing targets) could be abstracted to determine the approximate point when the targets are within range.

5.2.1.4 Numeric Representation

Another aspect of the model that can be abstracted is the representation of the system state. Consider the example of a model that is abstracted merely by representing all numeric values as integer values rather than double precision floating point values. Integer values logically

represent an approximation of the values represented in double precision representation. Since the number of bits used to represent the values in the computer is larger in double precision representation, converting the representation to integer values reduces the number of potential system states, which is abstraction by state aggregation. Further, one could consider state composition by defining an integer state to encompass all double precision states that round to the integer value. For example, the state $i = 1$ decomposes into the states whose double precision representation of the value of $i \in [0.5, 1.5)$.

5.2.2 Temporal Aggregation

The third type of aggregation in our taxonomy is defined as temporal abstraction. This type of abstraction reduces computational complexity by reducing the granularity of the time advance. This abstraction can be applied in either unit advance or event advance discrete event simulation. With unit advance, the abstraction technique is merely increasing the time step. With event advance, the abstraction involves assuming that events that occur "very closely" in time are considered simultaneous. Techniques described in Section 4.2.4 fit this category of abstraction technique.

Time warp time management schemes for parallel simulations (Jefferson 1985) are another example of temporal aggregation. The principle of time warp is to execute events in parallel, maintaining sufficient information to roll back the event sequences if the temporal ordering of events is significant to simulation outcome. In our terminology, this amounts to abstracting a set of states that differ only due to execution sequence into a state in which execution order is indeterminate. Rather than determining abstractions a priori, the time warp approach assumes that states can be abstracted, then reverts to the disaggregated state definition when it is determined that the disaggregated states contain significant distinctions.

5.2.3 Entity Aggregation

Abstraction by entity aggregation is a commonly used abstraction, particularly in models of military activity, such as those used by Rome Laboratory. Entity aggregation refers to the representation of a "higher level" entity to represent a collection of "lower level" entities. The use in military simulation is due to the inherent hierarchical structure of militaries that often drives simulation modeling requirements.

We note two different criteria by which entities are typically aggregated: structure and function. Each of these approaches is described below.

5.2.3.1 By Function

One approach to entity aggregation is to aggregate entities that perform a common or similar function. The resulting entity performs a function that is the aggregate of the individual functions. For example, consider a model of a bank in which there are multiple servers (tellers) but a single queue (line). A model could have each teller represented by a separate entity, or a

single aggregate entity (with correspondingly shorter service times) could be used. In this case the lower level entities are performing identical functions, and the aggregated entity is modeled as also performing that identical function.

Entity aggregation based on function can also occur when the individual entities are performing functions that contribute to the overall function of a higher level entity. For example, a detailed model of an aircraft in flight might have models for individual aircraft subsystems (propulsion, navigation, the aerodynamics of the aircraft body, and so on). Abstraction by entity aggregation results in a model in which the behavior of the aggregate model (the entire aircraft) is modeled in less detail.

5.2.3.2 By Structure

Entity aggregation can also be based on a hierarchical structure of entities, in which abstraction is accomplished by inserting higher level entities for lower level entities. For example, a model that represents the individual activities of a collection of tanks and armored personnel carriers could be replaced by a model of a tank company, or the behavior of individual aircraft could be replaced by behavior of a squadron of aircraft. Note that behavior in these examples is substantially different as a result of the abstraction, even though the model outcome is similar. The information required to model individual aircraft, and the behavior generated by such a model is much different than the information required by and generated by a model of an air squadron.

We distinguish this situation from entity aggregation by function. In this case the aggregated entity is performing different functions than the lower level entities, rather than a composite or union of lower level entity functions.

5.2.4 Function

The final technique for entity aggregation is that of function aggregation. Note that this is different from entity aggregation by function. Entity aggregation by function involves changing the entities that are represented in the model. Function aggregation does not change the entities that are represented, but aggregates the functions that those entities perform.

For example, consider an air warfare model that includes the activities required to prepare an aircraft for a new mission upon completion of a mission. Specific functions could be represented such as refueling, reloading weapons, damage repair, and so on. These functions could also be aggregated as simply a mission turn-around function. The aircraft and base entities still exist, but the details of some functions are abstracted.

5.3 Modification of Model Form

The third category of model abstraction techniques we refer to as abstraction by modification of model form. This category of abstraction techniques is characterized by a simplification of the input-output transformation within a model component. The same set of inputs is used in both the original and abstracted models, distinguishing this category from

abstractions by model boundary modification. The entities and time frames are the same, distinguishing this category from abstraction by behavior modification. The difference in the abstracted model is the manner in which a parameter value is determined. We use the term derived relationship because this abstraction technique is based on determining the input/output relationships and then using a more computationally efficient means to compute the parameter value.

Within this category, there are several subcategories of techniques for deriving relationships. The simplest technique that is often used by model developers is to replace a computation with a table of values. Another derived relationship is replacement of a particular deterministic computation with a random number drawn from a distribution that approximates the distribution of computed values. Analysis of input-output data relationships can also be analyzed to derive data for performing a linear function interpolation. In this case, the derived relationship is a discrete function represented by the values in a look-up table. The most sophisticated technique is known as metamodeling, in which the relationship between inputs and outputs of a model component is analyzed to derive a polynomial equation used to replace the component. The following subsections each describe one of these abstractions of model form in more detail.

5.3.1 Look-Up Table

The use of a look-up table is also a venerable tool among simulation model developers. In this abstraction, a transformation function is represented by a set of values. The input value is transformed into an index value into a data table, and the output value is then determined by retrieving the indexed value. In principle, it is similar to abstraction by probability distribution, in that a distribution of output values as a function of input values is derived. The look-up table requires some discretization of the input domain, but allows a broader set of functions to be represented than probability distributions. Look-up tables provide an efficient abstraction since they eliminate a substantial amount of computation; however, this approach requires increasing amounts of storage to increase the accuracy of the model.

5.3.2 Probability Distribution

The notion of replacing the computation of a parameter value with a pseudo-randomly-generated number has long been recognized as an abstraction technique. Zeigler (1976) identified it as a technique for creating lumped models. He identifies this technique as "replacing deterministic variables with random variables." For the purposes of this discussion, we ignore the fact that software implementations of random number generators are actually deterministic.

5.3.3 Linear Function Interpolation

Modeling by Linear Function Interpolation (LFI) provides a level of abstraction between that of a simple look-up table and a full polynomial representation. To increase resolution the classic look-up table requires additional data points. Doubling resolution requires doubling the

number of data points and so on. To model any given polynomial more accurately without increasing the number of breakpoints, a typical LFI system uses a look-up table whose entries are points (breakpoints) on the polynomial curve. A simple look-up table identifies the nearest tabulated value of the dependent variable to the input parameter and returns the corresponding independent variable value. By contrast, the LFI system identifies the two nearest breakpoints corresponding to the input parameter and returns an output value linearly interpolated between them in proportion to the proximity of the input variable to the breakpoints. Thus in effect the LFI model implementation serves to reduce a polynomial function to a series of straight line approximations.

5.3.4 Metamodeling

A metamodel is a mathematical approximation of a more complex process or model. It is derived from analysis of input-output pairs, generally treating the more detailed model as a black box. Early metamodeling work focused on developing a best polynomial fit for input/output data generated by the model. More recently, additional metamodeling techniques have been developed within the simulation community. Burton (1994) identifies the following categories of techniques in an overview of metamodeling techniques:

- parametric polynomial response surface approximations,
- splines,
- radial basis functions,
- kernel smoothing,
- spatial correlation models, and
- frequency-domain approximations.

This particular categorization is depicted in Figure 4 for completeness. We refer the reader to Burton (1994) for a discussion of, and references for, these metamodeling techniques.

These techniques share in common simpler approximations of the input-output transformations of a model. Metamodeling requires the existence of a more detailed model; it only operates in the direction of simplification. The work of Zeimer, Tew, Sargent, and Sisti (1993) is a recent example of the application of metamodeling to the types of C^3I models of interest to Rome Laboratory, specifically, the Tactical Electronic Reconnaissance Simulation Model (TERSM).

Caughlin (1994) also describes a metamodeling approach to identify the underlying processes that define the system. Caughlin extends the traditional metamodeling approach to incorporate knowledge of the system itself to identify what is essentially a reduced order model. By focusing on the latent variables of the model, the metamodel is not restricted to the data used

to build the model. Caughlin also demonstrates his metamodeling technique using the TERSM model.

5.4 Comparison with Other Taxonomies

The only other attempt to organize abstraction techniques is that outlined by Fishwick (1988) and summarized above in Section 3.2. Fishwick's taxonomy covers a broader scope, in that he includes items such as cerebral abstractions, which are beyond our specific focus on simulation models. However, his taxonomy does not address specific techniques; rather, his taxonomy is oriented towards the application of the abstraction (e.g., sensory presentation). Most of our techniques fit his broad definition of partial systems morphism (although we cite some examples of use of abstraction techniques for visual subsystems of real-time simulators, which falls into Fishwick's sensory abstraction category).

6. Application of Abstraction Techniques to the Reactive Wild Weasel Model

Having identified and categorized model abstraction techniques, we next turn to development and demonstration of those techniques. The purpose of this section is to document our investigation of applying model abstraction techniques to actual models used by Rome Laboratory. In this exercise we move from the more conceptual discussions of model abstraction techniques to more concrete examples. The purpose is to illustrate how these techniques could be applied in an operational setting.

The first model selected for this analysis is the Reactive Wild Weasel Model (RWWM). The RWWM simulates the target acquisition and weapons delivery capabilities of the F-4G Wild Weasel aircraft. The model was originally developed by Georgia Tech Research Institute, and later was integrated with other electronic combat models under the Electronic Combat Effectiveness Study (sponsored by Rome Laboratory). We selected the RWWM for the following reasons:

1. The model itself is relatively small (approximately 100 modules and 7K source lines of code), allowing us to concentrate on key concepts rather than wading through an extensive amount of code.
2. The model was previously used by Rome Laboratory under the Electronic Combat Effectiveness Study program, providing a relevant application; and
3. The authors of this report had previous experience with the model, reducing the effort required to learn about the model.

Limitations of the RWWM Study

Ideally, the most effective exercise using RWWM would have been to demonstrate abstraction techniques by modifying code and then comparing the results running the abstracted version of the model with the original version of the model. However, for the following reasons (generally due to the model's obsolescence) we were unable to carry this exercise through to code modification:

1. Not all of the files were available.
2. An initial walk through of the code identified several coding errors. While these errors could have been fixed, we were concerned that attempts to modify the model could run into other more subtle problems. We did not want this exercise to become bogged down in a RWWM debugging exercise.
3. No system level design documentation was available. This limitation had particularly significant implications for trying to modify critical modeling assumptions.

We limited this exercise to a "paper" analysis. In some cases we were able to use individual subroutines or code segments to illustrate and analyze the practical effects of an abstraction.

Despite the limitations, we were able to identify a number of potential model abstractions that could be applied to the RWWM.

We begin this section with a summary of the RWWM and a description of the process that was followed to extract model design information from RWWM to be used in the analysis of abstraction possibilities. This process is described in Section 6.1, with design tables and charts provided in Appendix B. We then describe how abstraction techniques could be applied to the model in Section 6.2. This section concludes with Section 6.3, which includes observations and lessons learned.

6.1 Summary of RWWM

The RWWM is designed to analyze equipment and weapons employment performing the mission of Suppression of Enemy Air Defense (SEAD). It simulates the F-4G Wild Weasel aircraft, receiver, crew, and weapons. It consists of two linked models: one of aircrew behavior, and one of the AN/APR-38 receiver. Aircraft fly pre-planned flight paths until reactions to the perceived environment generate deviations to accomplish Wild Weasel events. Aircraft and enemy weapon systems are considered destroyed at some defined level of probability of survival and are removed from the scenario. Inputs to the model include the following data files:

- Radar system data;
- Scan table processing data;
- Site data;
- Aircraft breakpoint data;
- Flight pairings data;
- Weapons/stores data;
- Aircraft weapons parameter data;
- Aircraft weapon versus threat P_k data; and
- Aircraft maneuver parameter data.

Outputs include a time history of the electronic order of battle (EOB), actual flight profiles, and aircrew decisions.

RWWM was designed to provide the AN/APR-38 EOB to the aircrew, implement the designated set of Wild Weasel aircrew rules of engagement, and determine the outcome of employing lethal suppression in various mission support roles. Analysis of which air defense weapon systems the Wild Weasel aircraft attack and destroy and which aircraft are engaged and destroyed are combined with the overall attack forces loss rate to quantify the relative contribution of lethal suppression.

The first step in analyzing the RWWM model was to understand the structure and operation of the model. In this respect we faced an interesting, though typical, challenge. Although the code was documented reasonably well, there was no system level documentation available to describe the fundamental assumptions and model design decisions on which the model was based. Thus our task was to perform some reverse engineering of the model code, constrained (as always) by the amount of effort available to perform the analysis. In this section, we present the process by which we gained sufficient familiarity to understand the model. This information is presented for two reasons. First, the results of this exercise will be useful for replicating or expanding upon this particular model abstraction exercise. Second, it may provide some useful ideas to anyone faced with a similar challenge of rapidly understanding a model without system level documentation. The resulting tables and charts are included as Appendix B of this report.

The first step was to establish the overall processing structure of the model. RWWM is structured in a manner typical of discrete event simulation models. There is an initialization step, in which both the model's exogenous variables and the initial conditions and initial event list are established. Then event list processing follows until the simulation clock reaches the user-input scenario end time. Event list processing consists of checking the event type from the event record stored in the event list, then executing the appropriate subroutines based on event type. These subroutines form the core functionality of the model, as well as the starting point in understanding how the model works.

The second step was to establish the "system boundary" of the model, as defined by the exogenous variables. A list was compiled of all inputs to the model (most of which were found in the initialization subroutine, INTIAL). Next, we determined whether the inputs represented exogenous inputs or the initial state of the system. **Exogenous variables** are distinguished by the fact that they are not modified in the model. The set of exogenous variables forms the **model boundary**, and abstraction techniques designed to modify the model boundary thus involve elimination of one (or more) of these parameters as a model input.

Next we needed to trace the processing associated with input variables. In order to understand the processing threads for the inputs and events, it was necessary to understand the organization of the code itself. We compiled a list of subroutines, including a brief description of the processing, and the calling sequence of the subroutines.

Having looked at the inputs and processing, the next logical step was to document the outputs of the model. There are several points at which the model outputs information about the current event being processed and parameters relevant to that event. These outputs document scenario events and aircrew decisions. Such information by itself would not generally meet our definition of a simulation requirement, since a single model execution only provides a single trajectory (of many possible) system states and behavior. In order to create meaningful abstractions, we hypothesized measures of effectiveness (MOEs) which could be generated as a result of multiple model executions.

In order to consider state aggregation abstractions, we also needed to identify the RWWM model variables that comprised the state definition. We began by considering variables that are initialized along with exogenous variables, but whose values are subsequently modified during the course of a simulation execution. A static analysis of the code can identify variables that may be modified, although for some scenarios a particular value may not be updated (i.e., an input is exogenous for a particular scenario). We limited our analysis to static code analysis, and we did not consider scenario-specific issues. Thus variables are categorized as exogenous variables only if there is no statement that updates their values; otherwise, variables are considered system state variables.

6.2 RWWM Abstraction Analysis

Having assembled the necessary design information, we next turn our attention to the system level abstraction opportunities. By system level, we are referring to abstractions that generally cut across individual subroutine and module boundaries, and reflect systemic boundary and behavioral changes. We divide this analysis according to the top level distinctions in the taxonomy of model abstraction techniques presented in Section 5. We begin by considering model boundary modifications (Section 6.2.1), followed by model behavior modifications (Section 6.2.2), and model form modifications (Section 6.2.3).

6.2.1 Model Boundary Modification

Techniques for modifying the system-level boundary are focused on elimination of the system level exogenous variables. In this section we discuss some ideas on how these parameters could be analyzed for potential abstractions. This discussion is organized by input parameter. For each parameter, the utilization of the parameter is defined, followed by an analysis of the impact of eliminating the parameter.

Radar System Parameters (RADSYS)

The RADSYS array contains a row for each radar site in the simulation, with seven parameters describing the radar site, as follows:

1. The **radar site identifier** value functions as a key for retrieving the other parameters for a particular radar site.
2. A **priority** value for the individual emitter that is added to a base priority value to determine the overall priority of an emitter for an aircraft. The base priority value is a function of emitter type, whether the Weasel is within critical range of the emitter, and other scenario dependent factors.
3. The **critical range** from emitter to airborne target. This parameter is used to determine whether an emitter is ignored or included in the Weasel's emitter prioritization scheme. In addition, the critical range is also used to determine

where a Weasel finishes an evasive maneuver (i.e., when it is beyond the critical range of the emitter) and whether a site can attack a Weasel.

4. The **target track db threshold,** which is used to determine target priority and to determine whether the ground site is engaging the Weasel.
5. The **missile guidance db threshold**, which is also used to determine target priority and to determine whether the ground site is engaging the Weasel.
6. The **launch db threshold**, which is used to determine a SAM site mode of operation, which in turn is used to determine target priority and whether the ground site is engaging the Weasel.
7. The **launch range threshold**, which is also used to determine a SAM site mode of operation, which in turn is used to determine target priority and whether the ground site is engaging the Weasel.

The radar site identifier is used as an index to the other values, and is not actually used as a parameter by the model algorithms. It would be eliminated if all radar site parameters were eliminated.

The priority parameter is a candidate for abstraction, as part of an abstraction of the computation of emitter priorities. This topic is discussed in more detail in Section 6.2.2.1.3.

Eliminating the critical range parameter would have a significant effect on model execution and most measures of effectiveness, since it dictates specifics of the Weasel flight path and whether ground-air engagement occurs. Abstraction of this parameter could only be implemented as part of a hierarchy of models that included a much simpler model of air-ground combat. Given the strong dependence of much of the model behavior (in terms of flight path and combat) on critical range, the relationship of model outputs for such a simplified model would be difficult (if not impossible) to predict.

The impact of the four threshold parameters is similar, since they contribute to the determination of whether the ground site engages the Weasel. Their contribution to the prioritization algorithm is scenario dependent. One approach to examining the abstraction possibilities here is to break the impact of these threshold parameters into two pieces: the impact of the parameters on the outcome of the prioritization algorithm, and the impact of the prioritization algorithm on the final model MOEs. The first question deals specifically with the individual parameters; the second question addresses a broader question of the significance of site prioritization.

Scan Table Processing Time (PT)

The scan table processing time parameters define the amount of time for each individual Weasel to scan each of its five EOB scan tables. Since events are scheduled in the event list based on completion of the scan, the main impact of this input is the frequency with which scan events

are executed within the scenario. Since this parameter is not directly incorporated into any mathematical formulation, its impact is not easily derived analytically.

Airspeed (AS)

The Airspeed array contains default airspeeds for four modes of aircraft operation: gather, attack, support, and evade. The airspeed of the aircraft itself is clearly an integral element of the simulation, and cannot be easily abstracted. However, it might be possible to abstract the four values into a single value, eliminating one of the distinctions among those behaviors. The significance of this abstraction depends in part on the relative separation of the airspeed values.

Turn Radius (TRNRAD)

The aircraft turn radius is used in the geometric computations of aircraft positions and maneuvers. It is one of the parameters that defines the maneuverability of the aircraft. The turn radius is integral to the flight path geometry, and abstraction of this value would reduce aircraft flight paths to sequences of straight lines.

As an expression of maneuverability, TRNRAD could be considered as a fitting parameter for the model. A turn radius of zero provides the greatest maneuverability; aircraft headings can be changed instantaneously (with first order discontinuity). For MOEs such as targets destroyed and aircraft survivability, this abstracted version of the model would overestimate (provide an upper bound for) the number of ground targets destroyed and underestimate (provide a lower bound for) the number of aircraft lost. A discussion of this abstraction of aircraft maneuverability is included in Section 6.2.2.1.4.

Turn Rate (TRNRATE)

The turn rate is used to determine where an aircraft is located in a turn, and when an end of turn event should be scheduled. The turn rate is an integral part of the overall model representation of aircraft maneuverability, so the considerations for abstracting the turn rate are largely those presented above for TRNRAD.

Although the turn rate and turn radius are related, it is possible to consider an abstracted model that ignores turn radius, but accounts for the turn rate. The effect of such a model would be to delay aircraft movement while changing headings without recomputing the aircraft location along the turns. Using TRNRATE as a fitting parameter is more difficult here because it is inversely related to maneuverability. An infinite turn rate represents greatest maneuverability, while the lower bound (TRNRATE=0) represents no maneuverability (i.e., the aircraft cannot change heading). A possible solution is to set the upper bound of the TRNRATE to 360° per minimum time increment as an upper bound, and consider a fitting approximation accordingly.

Pull Up Rate (PULRATE)

The pull up rate parameter is used to calculate launch angle and location for release of an AGM45 weapon. The calculation is performed as an iterative solution where the aircraft

position is projected along its heading as it pulls up to release the weapon. If the aircraft cannot pull up adequately before getting too close to the target, the launch is aborted.

There is the trivial abstraction of eliminating this parameter if no AGM45 weapons are used. More significantly, the parameter could also be eliminated by a simplified computation of launch point, which is described in Section 6.2.2.1.1.

Aircraft Mode (ACCODE)

The aircraft mode defines which of four fundamental behaviors are to be simulated for an aircraft. Since this parameter controls the behavior of the aircraft, it should only be eliminated in conjunction with a function aggregation that combines aircraft functions.

Pre-Planned Flight Path (PPFP)

The Pre-Planned Flight Path defines the planned course of each Wild Weasel. The determination of aircraft location in the model accesses this information for both routine movement along the path and resynchronization after maneuvers to launch aircraft or avoid attack. This information would be difficult to move outside the system boundary unless the entire model of aircraft movement was substantially abstracted.

Without eliminating the flight path information, it is possible to reduce the required computations by reducing the number of specific points in the flight path. Since the flight path information is still being used, this is not considered a modification of system boundary, but rather an aggregation of system states (see Section 6.2.2.1.2).

Wild Weasel Mission Type (WWMTYP)

The mission type distinguishes between certain behaviors (such whether the aircraft remains on-station or returns to base). For scenarios that focus on the execution of a single attack by a Wild Weasel against a single target, the behavior distinctions are not relevant, and this parameter could be eliminated. However, for simulation scenarios that involve aircraft activity over an extended period of time, mission type can impact MOEs such as number of targets destroyed.

Time Over Target (TOT)

The Time Over Target (TOT) parameter is part of the flight path definition, and used to constrain how long an aircraft has to attack its target before returning home (for some mission types). This parameter is fairly substantial in determining target destruction and aircraft survivability MOEs because ground sites can force the aircraft to take evasive action that results in a return to base before the aircraft can launch an attack. Without this restriction, aircraft could take an infinite amount of time to reach their target, and delaying tactics by the air defense sites would not be beneficial.

TOT is a bounded variable, in that it can not (or at least should not) be less than the start time of the scenario. We cannot assert that we know the outcome of the MOEs for TOT = t_0,

because the flight path and radar site geometry dictate what events occur before the TOT condition is checked. Intuitively, the relationship between TOT and number of ground targets destroyed is monotonic; the longer an aircraft has to reach targets, the more opportunity it has to successfully destroy units. TOT could be considered a fitting parameter, although like the case of TRNRAD, the precise relationship of the parameters cannot be derived. Arbitrary tuning of the fitting parameter, TOT, is not feasible. Note also that the relationship between TOT and aircraft survivability is indeterminate. The longer the aircraft is in the scenario, the greater the probability of aircraft attrition from active sites, while simultaneously, attrition of ground sites reduces the number of active sites that can destroy the aircraft.

USED

The USED parameter provides a constraint on the amount of fuel available to a Wild Weasel (different from its fuel capacity). The practical effect of this parameter is much like that of TOT, in that it limits the amount of time that an aircraft has to complete a mission. If this parameter were eliminated, then aircraft would only be limited to the fuel capacity values that are coded into the model software.

Because the practical effect of the USED parameter is to limit the amount of time aircraft cam function in the scenario, the preceding discussion of TOT as a fitting parameter applies to USED as well.

FLIGHT

FLIGHT establishes the relationship between a Wild Weasel and its wingman (if it has one). As such, it is used frequently in the model to update parameters of the wingman based on the activities of the lead aircraft. This information could only be eliminated in scenarios in which all aircraft operated independently.

Examination of the model code indicates that the current baseline version of RWWM already includes examples of abstractions. The wingman is assigned the same location and flight path parameter values, which is clearly an abstraction of the real-world separation of aircraft. Also, the model as currently written does not include a capability to separate (other than attrition) or join aircraft.[5] Thus an entity aggregation could be applied to the model, eliminating a separate aircraft by enhancing the capacity of the lead aircraft of a flight. This concept is discussed in Section 6.2.2.4.

STORES

The STORES array holds the weapons available for each aircraft during the course of the scenario. Unlike the other exogenous variables described in this section, STORES is updated as the scenario progresses to reflect the depletion of weapons during combat. There are several circumstances in which abstraction of this parameter would make sense. One model

[5]Such a capability was apparently envisioned, since there is a stub in the model for rejoining aircraft, but it has no executable code.

simplification is to not constrain the selection of the weapon; that is, assume that the best weapon is always available to the aircraft. This approach would yield an upper bound on the number of ground sites destroyed.

The issue of weapons selection would be irrelevant in scenarios in which the MOEs reflect the evasive capabilities of the aircraft. This approach suggests a dimension for abstraction that could be incorporated into a hierarchy of models. A simplified model that ignores offensive actions by the Wild Weasel could be at one level of the hierarchy, while the original model is at a more detailed level.

PST

The PST parameters are used in the computation of priorities that dictate the targets selected by the Wild Weasel for attack and the evasive maneuvers undertaken by the Wild Weasel. This parameter could be abstracted as part of a simplification of the priority computation scheme, as described in Section 6.2.2.1.3.

Weapon Probability of Kill (PKWEP)

The weapon P_k represents the probability that a type of weapon will destroy a radar site. Since it is a number in the range [0,1], it reduces the number of circumstances in which a radar site is destroyed. Elimination of the weapon P_k system input would result in weapons being completely effective. Thus such an abstraction would generate an upper bound MOE for destroyed ground targets. P_k could then be a fitting parameter, and abstracting the P_k would provide a bounded approximate output.

Weapon Release (WEPREL) and Weapon Range (WEPRNG)

These parameters are used to compute the geometry of an aircraft while launching a weapon. Abstraction of this parameter would only be applicable if the entire flight path model were abstracted.

Weapon Velocity (WEPVEL)

This parameter provides the temporal separation between weapons launch and weapon detonation. Without the temporal separation, it would be impossible for both an aircraft and a ground site to destroy each other. One entity would be destroyed before launching a weapon (even if the event times were truly simultaneous). The effect on the MOEs of such an abstraction cannot be determined, because it could result in either the ground site or the aircraft surviving in the abstracted model when both would be destroyed in the baseline model.

Position Change (POSCHG)

This parameter is used to compute the geometry of an aircraft after launching a weapon. Abstraction of this parameter would only be applicable if the entire flight path model were abstracted.

Site (S)

1: SYSTYPE.

2: Number of the scan table that contains the emitter.

3..5: Radar site coordinates.

6: Emitter signal strength.

7: Flag indicating whether emitter is active.

8: IPBE. Not used.

9: Emitter signal strength.

Radar site coordinates are fundamental to the model, since the location ultimately dictates the flight path of the Wild Weasel. The only way this parameter could be abstracted is as part of a major change to the level of model resolution that eliminated all location information, and only considered the number of sites in a given area.

The emitter activity flag is also fundamental to the model, since it indicates whether the emitter is emitting and can be detected. The deterministic sequence of emitter activity could be replaced with a random sequence of emitter activity. This modification would not simplify the model computation, but could reduce the effort to generate input data.

Emitter signal strength is used to determine whether an emission has sufficient strength to be detected, given the range from emitter to aircraft. Elimination of this parameter eliminates a critical distinction among types of emitters. If this distinction is not relevant, then a range threshold could be substituted for a signal strength threshold to simplify data input requirements.

TRACK

TRACK is a set of flags that indicate whether a particular radar site is tracking a particular Wild Weasel. That information is used to retrieve the appropriate signal db level. Elimination of the TRACK parameter is wholly dependent on how emitter signal strength is represented and whether the signal strength parameters can be abstracted.

6.2.2 System Level Modifications of Model Behavior

The second set of abstraction techniques are those which deal with behaviors within the model. In this section, we consider potential behavior abstraction possibilities, following the various approaches defined in the model abstraction taxonomy described in Section 5. The four major categories of behavior abstraction techniques include:

- State aggregation;
- Temporal aggregation;
- Function aggregation; and

- Entity aggregation.

Our discussion of candidate behavior abstraction is organized accordingly.

6.2.2.1 State Aggregation

State aggregation provides the broadest spectrum of potential abstraction techniques, since there is an infinite number of states to consider. For this reason, we utilize the analysis of the preceding section to provide some ideas of where to look for state aggregation abstractions. In that discussion, we noted four situations in which we could eliminate an input parameter as a secondary abstraction to a primary state aggregation. These four abstractions involve:

- Simplification of the launch point computation for the AGM-45;
- Reduction in the number of flight path parameters;
- Simplification of the priority computation; and
- Simplification of aircraft maneuvering.

Each of these topics is addressed in a section below.

6.2.2.1.1 Launch Point Computation

The launch point computation in the subroutine EX45 for the AGM-45 weapon provides a useful opportunity exploring different abstraction approaches, including approximation of level flight (elimination of the pull up rate parameter) and aggregation of repeating cycles. These two abstractions are each described in a subsection below. First we summarize the existing launch point calculation in RWWM, then discuss approaches to abstracting that component of the model.

EX45 is called by module PULUP when a detected site is to be attacked using an AGM45 weapon. The range of this weapon after release from the aircraft is a function of aircraft pitch at launch and altitude, plus a constant roughly equivalent to 4.2 nautical miles. The module initially assumes zero launch pitch and current altitude and then calculates weapon range. If weapon range is more than 1000 feet short of the current distance to target, the module iteratively adjusts the launch parameters and recomputes new weapon ranges until the weapons range is within 1000 feet of the range to target. This is in essence a "what if" calculation in which all the iterations are completed during a single subroutine call. The adjustment is iteratively calculated as follows (actual parameter names are shown in parentheses):

1. An increment is applied to the weapon launch angle (LANG) proportional to the nominal aircraft pull-up rate (PULRATE) applied for 0.1 seconds.
2. The horizontal distance (DELTAR) the aircraft would travel while reaching that pitch angle, and the gain in altitude, are calculated.

3. The distance traveled and altitude gained are used to predict: a revised distance to target (APRRNG); the distance of launch point from present location (R); and the launch altitude (ALT) that would result from use of the revised launch angle.

4. Weapon range (XRNG) is recomputed using these new values, and compared with the new (and shorter) distance to target. The cycle is repeated until the weapons range is within the acceptable envelope, or until acceptable pitch limits are exceeded, in which case the calculation is aborted.

5. When the range conditions are met, the pitch, altitude, and distance to launch point data are output.

If the weapon range at zero launch pitch is greater than the distance to the target, and the weapon would therefore overshoot the target, the module performs a similar calculation but decrements the launch pitch angle. This implies that at this point in the simulation the aircraft altitude is above some nominal value which allows for a dive, since there is no protection to preclude a dive from low altitude.

The model FORTRAN code (taken from EX45) for this algorithm is as follows:

```
LANG = 0.0
XRNG = 1.789*ALT + 986.*LANG + 25766.22
DO 10 I = 1, 95
         IF ( (XRNG .GT. (APRRNG-1000.)) .AND.
     1     (XRNG .LT. (APRRNG+1000)) ) THEN
             GO TO 11
         ELSE IF ( XRNG .LT. (APRRNG-1000.) ) THEN
            LANG = LANG + PULRATE * 0.1
         ELSE
            LANG = LANG - PULRATE * 0.1
         END IF
***
***      * CHECK TO SEE IF LAUNCH ANGLE IS OUT OF PARAMETERS AND
***      * IF SO DROP OUT OF LOOP AND ABORT.
***
         IF ( (LANG .LT. -20.) .OR. (LANG .GT. 45.) ) THEN
            ABORT = .TRUE.
            GO TO 11
         END IF
         ALT = ALT + (SIN(LANG/57.3)*AIRSPED*0.1)
         DELTAR = COS(LANG/57.3)*AIRSPED*0.1
         APRRNG = APRRNG - DELTAR
         R = R + DELTAR
         XRNG = 1.789*ALT + 986.*LANG + 25766.22
         DELTAT = DELTAT + 0.1
  10  CONTINUE
  11  CONTINUE
```

For the purpose of this discussion, we express the algorithm in mathematical terms as follows:

We need to find T such that:

$$|w - r| < 1000$$

where

$$w = 1.789 * a + 986 * \deg(l) + 25766.22$$

(where deg(x) indicates a function to convert radians to degrees), and:

$$l = pT$$

$$a = a_0 + \sin(l) * sT$$

$$r = r_0 + \cos(l) * sT$$

where

- a_0 = Aircraft starting altitude (initial value of ALT)
- a = Aircraft altitude
- p = Pull up rate (PULUP)
- l = Current launch angle (LANG)
- T = Time (DELTAT)
- s = Airspeed (AIRSPED)
- w = Weapon range (XRNG)
- r = Range to target (APRRNG)
- r_0 = Initial range to target

6.2.2.1.1.1 Abstraction by Level Flight Approximation

One simple abstraction of this module is elimination of the PULUP parameter. This represents the situation in which weapons are launched from level flight. The computation becomes straight forward; the weapons range is calculated based on the aircraft altitude (which is now constant). The distance required for the aircraft to be able to deliver the weapon within 1000 feet of the target can immediately be determined. The simulation time for the weapon launch is determined by dividing this distance by the aircraft's airspeed.

The impact of this abstraction depends upon the initial situation. In cases where the aircraft begins beyond the weapons delivery range, the effect of the approximation is to require the aircraft to be closer (and lower) to the ground site, increasing the probability of aircraft attrition. In situations in which the aircraft is too high or too close to the target, the effect of the approximation is to eliminate an opportunity to attack the target, reducing the probability to target destruction.

Implementation of this approximation involves computing the time to launch the weapon as follows. Since we are calculating a specific point, the boundary condition is expressed as an equality: $|w-r|=1000$, where

$$w \approx 1.789a_0 + 25766.22$$

(the 986*deg(l) term is removed as the implementation of the abstraction), and

$$r \approx r_0 - sT$$

Thus T can be calculated simply as

$$T \approx \frac{r_0 - 1.789a_0 - 25766.22 \pm 1000}{s}$$

6.2.2.1.1.2 Abstraction by Aggregating Repeating Cycles

We can abstract the launch point computation in a more elegant way which allows the pull up rate to be incorporated into the model without significantly more computation. The iterative solution used in RWWM involves a cycle of very similar behaviors. The system states at the end of each cycle are not identical, because the aircraft altitude and range to target change. However, the states are similar in that the aircraft is still moving towards the target with the objective of moving within range of its weapon to be delivered. The algorithm in the model can be replaced by aggregating these cycles into a single cycle in which the aircraft moves to the weapons release point.

We begin by substituting the continuous expression for the altitude. Thus:

$$a = a_0 \pm \int_0^T s * \sin(pT)dt$$

or

$$a = a_0 \pm \left(\frac{s}{p}(1-\cos(pT))\right)$$

The minus sign reflects the possibility that the aircraft may need to decrease altitude to get within range. For simplicity, the remainder of this discussion deals with the case of increasing altitude. Decreasing altitude can be handled in much the same way. Note that although we are in the process of defining an abstraction that will approximate the results of this algorithm, the preceding expression for a is actually more accurate than the original model. In EX45 the change in altitude is calculated as simply the product of speed, time, and sine of final pitch angle for the iteration. Since pitch angle is constantly changing over the iteration, the integral generates a more accurate answer. The same argument applies to the range delta calculation.

Substituting these expressions for l and a back into the original equation for w yields:

$$w = 1.789a_0 + \frac{1.789s}{p}(1-\cos(pT)) + 986\left(\frac{180}{\pi}\right)pT + 25766.22$$

Range is the horizontal component of the launch angle, defined as:

$$r = r_0 - \int_0^T s\cos(pT)dt$$

which reduces to

$$r = r_0 - \frac{s}{p}\sin(pT)$$

Substituting back into the original equation, we must solve for T such that:

$$\left|1.789a_0 + \frac{1.789s}{p}(1-\cos(pT)) + 986\left(\frac{180}{\pi}\right)pT + 25766.22 - \left(r_0 - \frac{s}{p}\sin(pT)\right)\right| = 1000$$

All parameters except T can be treated as constants in this calculation, so we move all of the constant terms to one side of the equation, leaving:

$$-\alpha\cos(pT) + \beta\sin(pT) + \gamma pT = \kappa$$

where:

$$\alpha = \frac{1.789s}{p}$$

$$\beta = \frac{s}{p}$$

$$\gamma = 986\left(\frac{180}{\pi}\right)$$

$$\kappa = r_0 - 1.789a_0 - \frac{1.789s}{p} - 25766.22 \pm 1000$$

Over the range 0° to 45°, the sine and cosine terms of the above equation taken together can be approximated by a straight line from (0,-α) to (π/4, βsin(45°)-αcos(45°)). More specifically, we use the equation for this straight line approximation and substitute:

$$\frac{\frac{\sqrt{2}}{2}(\beta-\alpha)+\alpha}{\frac{\pi}{4}}pT - \alpha$$

for the sine and cosine terms, yielding the following approximate expression for T.

$$T \approx \frac{\kappa + \alpha}{p\left(\gamma + \frac{\frac{\sqrt{2}}{2}(\beta - \alpha) + \alpha}{\frac{\pi}{4}}\right)}$$

Expressed in terms of the original values, T is approximated as

$$T \approx \frac{r_0 - 1.789a_0 - 25766.22 \pm 1000}{\frac{986 * 180}{\pi} + \frac{1.789 - \frac{\sqrt{2}}{2} * (.789)}{\frac{\pi}{4}} s}$$

A more detailed step by step derivation of this result is provided in Appendix C.

6.2.2.1.1.3 Comparison of Abstractions

The two preceding sections provided two different methods for approximating the launch weapon release time (along with altitude and launch angle, which can be derived from the launch time). In this section, we compare the results of these two approximations with the results of the original model.

Figures 7 through 11 portray the results of this abstraction for a variety of scenarios. In each case, the RWWM model algorithm and the abstracted version were executed with a nominal set of initial parameters, as follows:

Airspeed: 250 nm/hr

Pull-up rate: 1°/sec

Starting altitude: 5000 ft

Starting range to target: 60000 ft

In each chart, one of these parameters is varied to show how the abstractions and the actual model determined the launch time as a function of these parameters.

Figure 7 shows the computation of launch weapon time with airspeed, pull up rate, and starting altitude values shown above, and the starting range to target varying from 105000 feet down to 40000 feet. Since the nominal pull up rate is 1° per second, the values of the launch time on the Y axis also equate to the launch angle (except for the level flight approximation, in which the launch angle is always 0°). This particular set of input values generates launch angle outputs across almost the entire range of allowable values [0°,45°].

Three curves are shown in Figure 7: the launch times generated by the original RWWM model, the launch times generated by the level flight abstraction described in Section 6.3.2.1.1.1, and the aggregating cycles abstraction described in Section 6.3.2.1.1.2. Clearly the level flight

approximation significantly overestimates the time to launch. At a pull-up rate of 1° per second, the weapons range in the original case increases by 986 feet per second, which is more than twice the ground speed of the aircraft in this example. Elimination of this term in the level flight approximation is responsible for the large variance between the approximation and the original model. The magnitude of the variance between the level flight approximation and the model results depends on the scenario parameters. For example, as airspeed increases, the level flight approximation results move closer to the original model results (see Figure 9). Similarly, this variance decreases with smaller pull up rates (see Figure 11), and this variance decreases as aircraft airspeed increases.

The approximation by aggregating cycles is a much closer approximation, although the shapes of the curves are significant. To illustrate, we depict the same curves on a larger scale in Figure 8. Note that in this figure the approximation is closest to the output of the original model at the extremes of the possible values of T. This is consistent with our straight-line approximation of the sine-cosine term of the approximation, which was computed based on the values of pT at 0° and 45°.

Figures 9 and 10 show similar graphs of the outputs of the three versions of the model, with variations in airspeed and altitude. Here again the trend is the same: the level flight approximation substantially overestimates the launch time, while the aggregated cycles slightly underestimates the launch time.

Figure 11 contains the model outputs for varying pull-up rates. Since this parameter is eliminated in the level flight approximation, that output is constant for all test cases.

The aggregation of cycles consistently underestimates the time required for the aircraft to get within range. This is based on the approximation used in the substitution for the sin(pT) and cosine(pT) terms of the equation. We can exploit this fact by aggregating the cycles up to the approximate time, then follow the original algorithm. This approach results in the abstraction yielding the identical results as the original model, with less computation.

It is possible for the aggregation of cycles abstraction to generate a negative time. The significance of a negative result is that the aircraft cannot delivery the weapon within range, typically because the aircraft is too high and close to the target. In this situation, the same sequences of cycles is performed in the original model, except that the aircraft is decreasing altitude rather than increasing altitude. A similar abstraction approach can be followed as well.

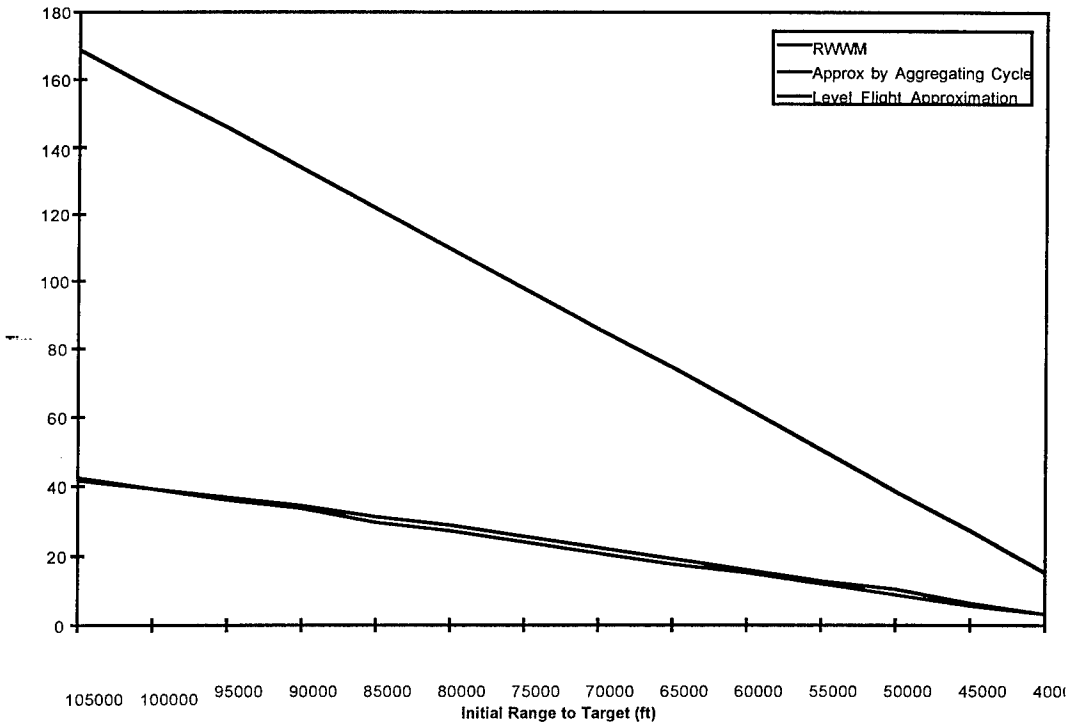


Figure 7, Computation of Weapon Launch Time with Varying Initial Range

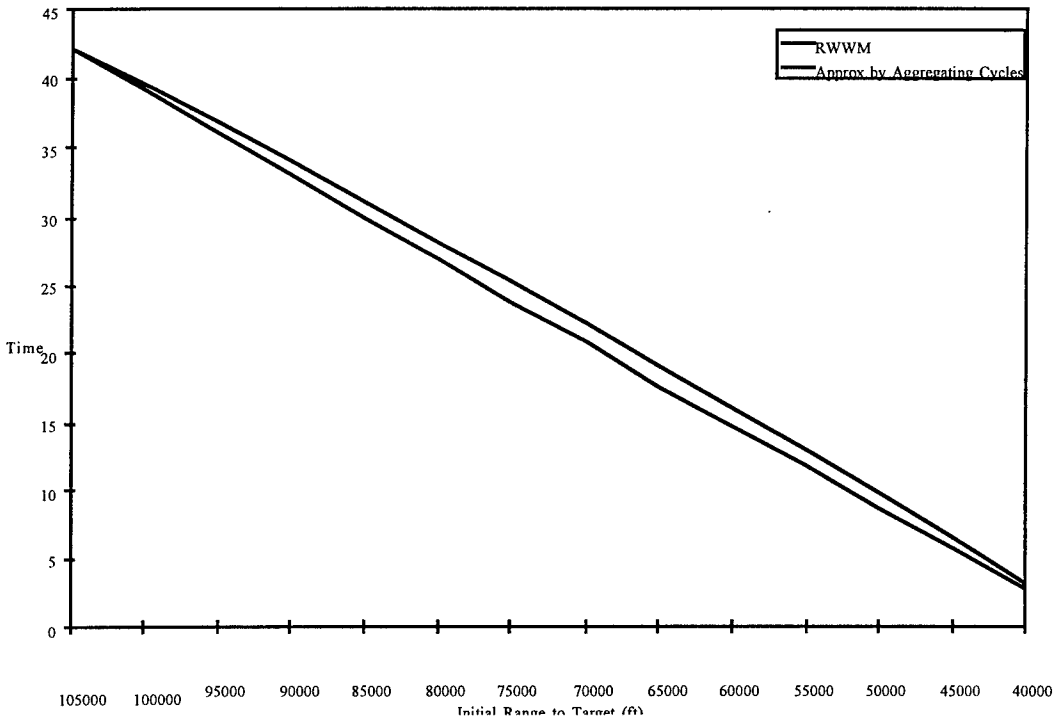


Figure 8, Computation of Weapon Launch Time with Varying Initial Range (Expanded View)

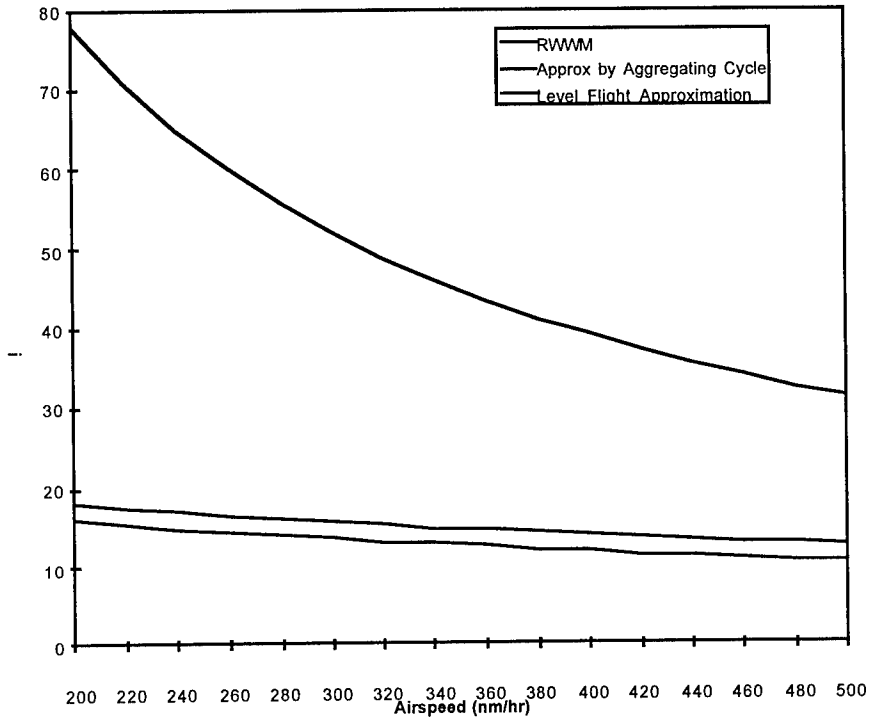


Figure 9, Computation of Weapon Launch Time with Varying Airspeed

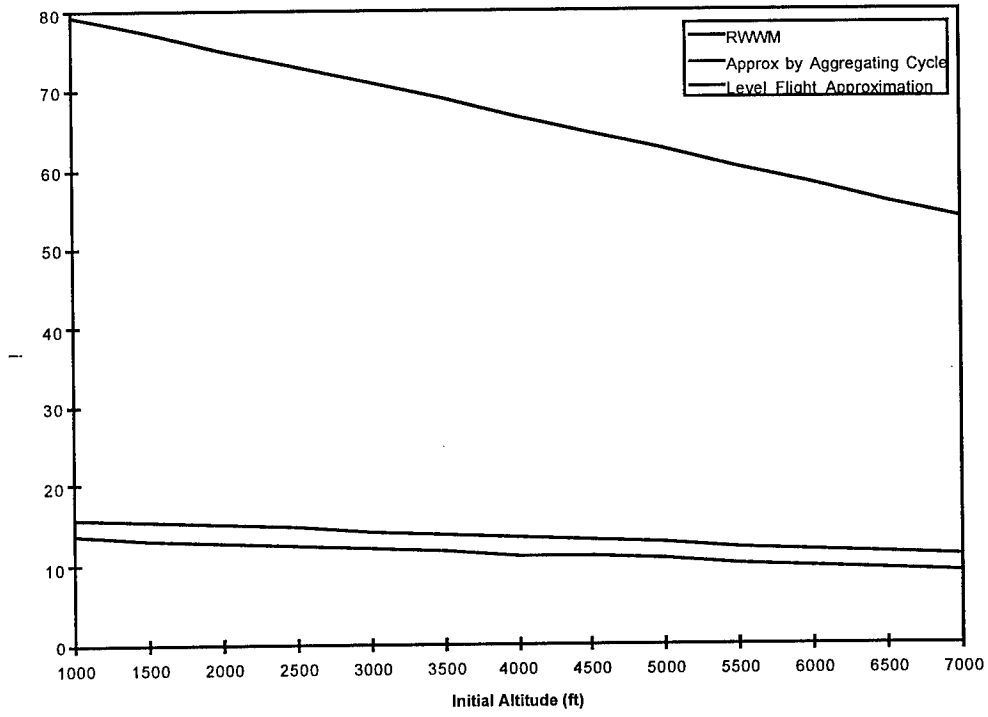


Figure 10, Computation of Weapon Launch Time with Varying Initial Altitude

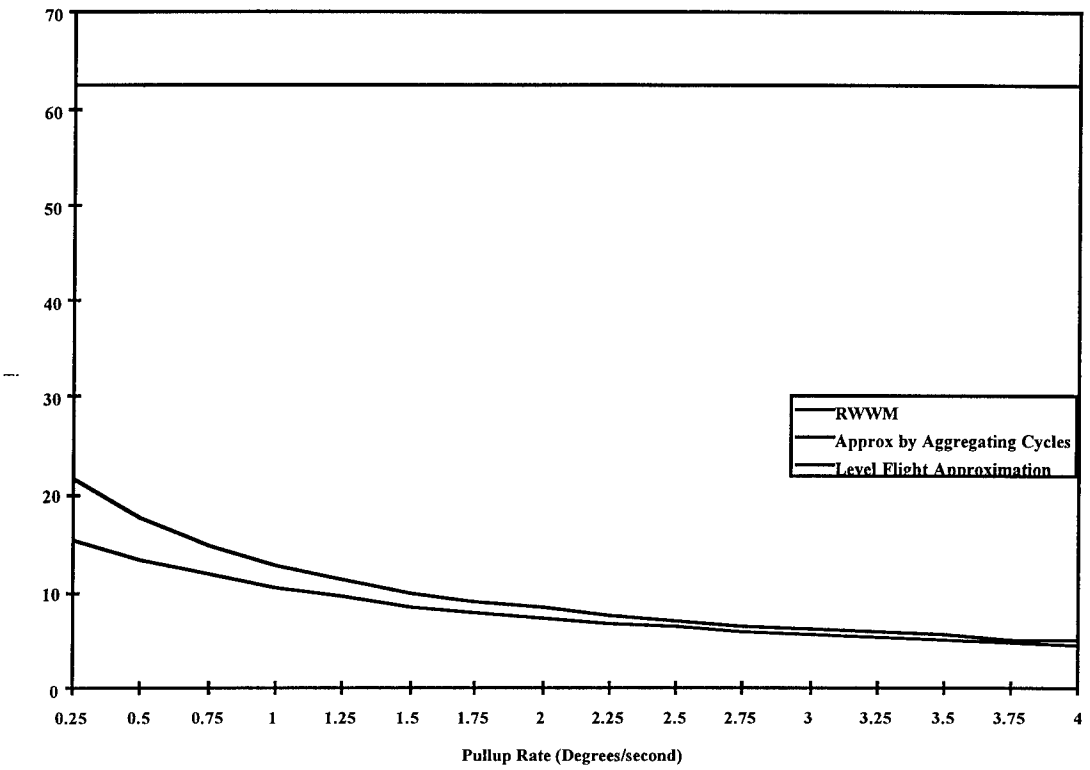


Figure 11, Computation of Weapon Launch Time with Varying Pull-up Rate

6.2.2.1.1.4 Resolution Abstraction

The iterative approach used in EX45 can also be abstracted by considering the terminating conditions. The calculations terminate when launch parameters are determined which give a weapon range within 1000 feet of the target distance. Thus any aircraft location in which the weapon delivery distance is within 1000 feet of the target range is treated as a the same state. This nominal accuracy could be set to a larger value, representing the aggregation of multiple states into a single "close enough" state. The computation terminates after fewer iterations. A simple test case was generated and executed to explore this approach, in conjunction with temporal aggregation. The results of this exercise are presented in Section 6.3.2.2.

6.2.2.1.2 Flight Path Point Aggregation

The aircraft in the model move along flight paths that are defined in terms of a series of break points. The arrival of an aircraft at a break point is a significant event in the model. A pre-planned flight path is defined as part of the input to the model. One approach to abstracting the model is to reduce the number of break points. Since this serves to reduce the number of potential system states, it is an example of state aggregation.

6.2.2.1.3 Priority Computation

The priority computation is used to determine which emitters are the greatest threat to, or most important targets for, a Wild Weasel aircraft. The priority computation algorithm is summarized in Table 1. In this section we consider approaches for abstracting this component of the model. The algorithm determines a priority based on six different state parameters. Some of the variables are essentially binary, while others (such as radar type) can assume one of several values. As shown in Table 1, there are forty-five (45) possible results, i.e., there is a quantization of forty-five levels of priority. One simplification is to reduce the number of discrete priority values by reducing the number of parameters used to determine the priority. The rationale for implementing this abstraction is the impact on the MOE. While changing the priority computation will change the behavior of the aircraft, the significant issue is the extent to which it changes the overall outcome.

The significance of the input site priority parameter P is based on the relative order of magnitude of the parameter values. Base priorities that are a function of the emitter type and system state parameters (such as relative range, relative azimuth) are generally separated by 100 units. If this input site priority parameter is less than 100, then the effect of site-specific parameter is to break ties among emitters that are in an otherwise identical state (at a particular point in time). On the other hand, if the site priority parameters are values generally greater than 5000, that site priority value dominates. The priorities assigned on the basis of the dynamic state would then serve to break ties among the individual emitters. Assuming that the former situation is the case, the prioritization is a logical candidate for abstraction. If the emitter type and battlefield dynamics do not provide a distinction as to which site should have a higher priority, then the distinction provided by the priority of individual emitters will not have a significant impact on typical MOEs. Implementing this abstraction involves eliminating the input parameter and eliminating the addition of P in the priority computation algorithm.

Table 1, Site Priority Computation Algorithm

SITE PRIORITY COMPUTATION ALGORITHM						
Radar type	**SAM radar mode of operations**	**Track flag**	**Range quality**	**Range**	**Dotted parameter[6]**	**Priority =**
< 200						5000 + EW
= 201		= 1	≤ 2	≤ 5		400
				> 5		4300
			> 2			800
		≠ 1	≤ 2	≤ 5		1300
				> 5		4500

[6]Number of scan cycles site has or has not been on the air.

SITE PRIORITY COMPUTATION ALGORITHM						
Radar type	**SAM radar mode of operations**	**Track flag**	**Range quality**	**Range**	**Dotted parameter[6]**	**Priority =**
			> 2			1700
(201, 300)		= 1	≤ 2	≤ 5		500 + P
				> 5		4600 + P
			> 2			900 + P
		≠ 1	≤ 2	≤ 5		1400 + P
				> 5		4700 + P
			> 2			4100 + P
< 400	= 0	= 1	≤ 2	≤ CR		3600 + P
				> CR		4200 + P
			> 2			3700 + P
		≠ 1	≤ 2	≤ CR		3900 + P
				> CR		4400 + P
			> 2			4000 + P
	= 1	= 1	≤ 2	≤ CR	≥ 10	1900 + P
					< 10	2000 + P
				> CR	≥ 10	3100 + P
					< 10	3200 + P
			> 2		≥ 10	2300 + P
					< 10	2400 + P
		≠ 1	≤ 2	≤ CR	≥ 10	2500 + P
					< 10	2600 + P
				> CR	≥ 10	3300 + P
					< 10	3400 + P
			> 2		≥ 10	2700 + P
					< 10	2800 + P
< 400	> 1	= 1	≤ 2	≤ CR	≥ 10	200 + P
					< 10	300 + P
				> CR	≥ 10	2100 + P
					< 10	2200 + P
			> 2		≥ 10	600 + P
					< 10	700 + P
		≠ 1	≤ 2	≤ CR	≥ 10	1100 + P

SITE PRIORITY COMPUTATION ALGORITHM						
Radar type	**SAM radar mode of operations**	**Track flag**	**Range quality**	**Range**	**Dotted parameter[6]**	**Priority =**
					< 10	1200 + P
				> CR	≥ 10	2900 + P
					< 10	3000 + P
			> 2		≥ 10	1500 + P
					< 10	1600 + P
> 400		= 1				100 + P
		≠ 1				1800 + P

where:

P is the input priority value for the specific site
CR is the critical range for the specific site
Radar type is stored in the site EOB array SEOB (i,j,1)
SAM radar mode of operations is stored in the site EOB array SEOB (i,j,12)
Track flag is stored in the site EOB array SEOB (i,j,13)
Range quality is stored in the site EOB array SEOB (i,j,14)
Range is stored in the site EOB array SEOB (i,j,5)
Dotted parameter is stored in the site EOB array SEOB (i,j,15)
Priority is the computed priority written into the site EOB array SEOB (i,j,17)

Other parameters could also be eliminated. As seen in Table 1, the dotted parameter contributes the least significant information toward the priority computation, providing a tiebreaker when other site parameters are the same. Thus elimination of this parameter would likely cause the least variation from the baseline model.

This type of abstraction is an example of state aggregation. The priority assignment is based on distinctions among sites. Simplifying the priority computation by eliminating a parameter is eliminating a distinction that is irrelevant to the final outcome.

6.2.2.1.4 Maneuver Computation

The maneuver computation abstraction is based on simplifying the computation of the flight path in the model. RWWM has a substantial amount of code which determines the flight path for the aircraft when it deviates from its predefined flight path. A significant constraint in determining the flight path is the turning radius of the aircraft. For example, in the subroutine GETBCK, the flight path is computed as one or more arcs of minimum radius TRNRAD necessary to maneuver back to the pre-defined flight path at a pre-defined point and heading.

A simplified version of this computation could be implemented by eliminating the turn radius constraint. In the specific case of GETBCK, the computation, which includes calls to

several subroutines, would be replaced by an immediate change to a heading to the appropriate point on the pre-planned flight path.

One of the issues in implementing this type of abstraction is to ensure that the relationship between the MOEs and the fitting parameter (TRNRAD) is monotonic. Intuitively, maneuverability, as expressed by the parameter TRNRAD, serves to constrain flight paths, which provides the Wild Weasel with fewer options for avoiding attacks from radar sites or pursuing them as targets. To ensure the preservation of the output validity, we must conclusively establish that the relationship is monotonic. One approach is to prove the relationship logically or mathematically, as is done in qualitative simulations, as described by Weld (1992). However, in this model the relationship between maneuverability and the MOEs is not nearly as direct as models expressed as mathematical equations. Further, the MOEs are affected by other factors whose influence is difficult to assess. For example, greater maneuverability could allow the Wild Weasel to attack more targets, resulting in higher ground target kills but also higher aircraft attrition. The specific relationship between MOEs and TRNRAD is also a function of the scenario-specific geographic distribution of radar sites. The more closely spaced the radar sites, the more significant the role of TRNRAD. Thus a specific statement about the impact on TRNRAD on MOEs cannot be made.

The alternative approach to validation of the abstracted model would require analysis of data generated by the abstracted model. This approach requires running a number of test cases varying the TRNRAD (under a variety of scenario conditions) and examining the impact on the MOEs. This data could also be used to derive a metamodel of RWWM. The metamodel then shows the relationship between TRNRAD and the MOEs, including whether that relationship is monotonic.

There is one other key aspect of this application that differs from the concept of Weld's approximation approach using model sensitivity analysis. Because the relationship of TRNRAD to MOEs is not well defined, it is impossible to truly "tune" TRNRAD to an arbitrary tolerance, although a metamodel or information derived from model runs could be useful.

6.2.2.2 Temporal Aggregation

Since RWWM is designed for sequential processing and uses an event advance time mechanism, system level temporal abstraction cannot be effectively applied. Defining larger time units such that ordering of events within the time envelope becomes irrelevant will not reduce the computational requirements of the model.

However, the EX45 subroutine which we considered for aggregation of repeated cycles also provides an opportunity to look at temporal aggregation as well. As noted during our analysis of the function of this module, the iterations through the calculations are conducted for temporal increments of 0.1 seconds. Adjustment of the size of the time increment changes the number of iterations, and hence can some save computational time. A limit exists on how much less

stringent the time increment can be made since the resultant weapon range increment must not cause weapon range to overshoot the target distance plus or minus 1000 feet.

To illustrate this example of temporal aggregation, we recoded and implemented the EX45 module in C. The original FORTRAN code was not used due to the absence of supporting data files and the presence of source code errors. A code segment was written in C and used to generate several data sets. This program was written with a translation of EX45 embedded within it. The C program also contained the code needed to initialize all parameters that would have been calculated by other routines prior to access by EX45. Reasonable test values were set for parameters such as aircraft altitude and airspeed, missile site number, and so on, using the same input array format as used in the original model. The capability was included to specify different RWW aircraft, with different values for these parameters. A listing of the C program is provided in Appendix E.

The program was also set up with a loop designed to execute the EX45 copy using a variety of iteration increments. For each of these permutations of the model, the following parameters were output: the distance to target from the weapons release point; the launch angle at weapons release point; and the number of iterations needed to arrive at that conclusion. The results generated by the program for a typical set of these variations are shown in Table 2.

For any given selection of iteration interval, the results shown in Table 2 confirm the expected results. The data indicates a dependency between the required accuracy and the number of iterations of the program to reach that accuracy. Similarly, for any given selection of required accuracy, a correlation exists between the iteration interval and the number of iterations needed. An interesting anomaly does occur for the combination of large iteration interval and small allowed miss distance, where a total of 14 iterations are required to reach a solution. This phenomenon is apparently caused by repeated overshoot of the target distance (plus or minus x feet) by the calculated weapon range, which results in excessive iterations.

A second nested loop was also provided so that a range of weapon allowed accuracies was considered for each of the iteration increment values. This corresponds to resolution abstraction variations suggested by Section 6.3.2.1.1.4 above.

For an "allowed miss" of 1000 feet, the same result is generated for iteration intervals up to 0.2 seconds, although with different numbers of iterations. In this particular case, iteration intervals of greater than 0.2 seconds result in the launch point determination after the aircraft has passed the point where weapon range and range to target are identical. Thus in this particular case an interval of 0.2 seconds is best. In other words, a temporal aggregation which aggregates two time intervals (two intervals of 0.1 second duration) could be applied in this particular case.

Table 2, EX45 Temporal Aggregation Results

EX45 TEMPORAL AGGREGATION RESULTS

Iteration Interval	Allowed Miss	Actual Miss	Launch Angle	Iterations Needed
0.025	250.0	-191.6	1.40	17
0.025	500.0	-483.4	1.14	14
0.025	1000.0	-969.1	0.70	9
0.025	2000.0	-1744.8	0.00	1
0.050	250.0	-191.4	1.40	9
0.050	500.0	-386.0	1.22	8
0.050	1000.0	-969.0	0.70	5
0.050	2000.0	-1744.8	0.00	1
0.100	250.0	-191.0	1.40	5
0.100	500.0	-191.0	1.40	5
0.100	1000.0	-968.8	0.70	3
0.100	2000.0	-1744.8	0.00	1
0.200	250.0	-190.0	1.40	3
0.200	500.0	-190.0	1.40	3
0.200	1000.0	-968.3	0.70	2
0.200	2000.0	-1744.8	0.00	1
0.400	250.0	-188.2	1.40	2
0.400	500.0	-188.2	1.40	2
0.400	1000.0	-188.2	1.40	2
0.400	2000.0	-1744.8	0.00	1
0.800	250.0	-148.2	-2.80	14
0.800	500.0	-335.9	-0.00	5
0.800	1000.0	-335.9	-0.00	5
0.800	2000.0	-1744.8	0.00	1

Aircraft = WW#1:
AIRSPD = 250/3600 miles/second
Range = 6 miles
Altitude = 5k ft

6.2.2.3 Function Aggregation

Identification of function aggregation opportunities requires a detailed understanding of the conceptual basis of the model. An aggregated function requires a fundamental change in the manner in which the real world system is represented. For example, the ACCODE parameter defines one of four modes of aircraft operation: gather, attack, support, or evade. Function aggregation would involve combining some of these functions together on the basis that the distinctions in performing this function are not relevant to the simulation output. In the RWWM example, intuition suggests that evade and attack are sufficiently different functions that a higher level aggregation would be difficult to define. On the other hand, an aircraft in attack mode performs the same function as an aircraft in gather mode until the aircraft is within 60 seconds of the intercept point. This commonalty could form the basis for an aggregation function.

Another approach to analyzing function aggregations is to review the definition of the event types, since these represent a high level description of the functions of the simulated entities. The events in RWWM are generally tied to the maneuvering of the aircraft or are fundamental events for the ground sites. Aggregating maneuver functions would require reconsideration of the level of aircraft flight fidelity (in much the same manner as eliminating the turning parameters). However, without the RWWM design documentation to better clarify the aircraft flight modeling assumptions, it is difficult to determine the applicability of potential function aggregations.

6.2.2.4 Entity Aggregation

One example of entity aggregation that is feasible given the current construction of the model involves the representation of wingmen. Although it appears from the in-line documentation that the eventual goal of the model was to model aircraft separating and rejoining, the current version does not include such a capability. As a result, pairs of aircraft designated as a flight are not separated except as the result of attrition, in which case the remaining aircraft must continue alone. Pairs of aircraft are given the identical flight path, but maintain different fuel levels, weapons, and attrition. Aircraft modes (designating which aircraft is attacking and which is supporting) may change during the course of the scenario.

Although not computationally intense, there are many points in the model where the existence of a wingman must be checked, and its information must be updated accordingly. This situation lends itself to an entity abstraction. The same model outputs can be generated by aggregating the lead and support aircraft into a single entity, eliminating the need to update the second aircraft. The parameters of the aggregated entity would be determined based on an appropriate combination of the individual entity parameters. For example, the weapons of the aggregated entity would be the union of the weapons of the individual entities, but the available fuel would be the minimum of the fuel available on the individual aircraft. Some additional information would need to be maintained in the model for this aggregation. For example, incremental attrition would need to be maintained to reflect the situation when one of the two aircraft was destroyed. On the other hand, some simplification of the target selection and

weapon assignment logic could be implemented, since there would be only one aircraft entity to consider. On balance, the aggregated model would be simpler without changing the resultant MOEs.

6.2.3 System Level Derived Relationships

In Section 5.3, we define this category of model abstraction as the simplification of the input-output transformation within a model component. The input-output relationship is determined, and a method that is more computationally efficient than the original model is then substituted to achieve that same relationship. The category is also subdivided into Look-Up Table, Probability Distribution, Linear Function Interpolation, and Metamodeling approaches.

There are places within the RWWM where derived relationships are already in use. For example, the detailed determination of whether a target is destroyed is replaced with a probability distribution. Since the use of look-up tables, probability distributions, and linear function interpolation is frequently used within the C^3I modeling and simulation domain, we skip detailed discussion of their application in this context. Likewise, the application of metamodeling has been investigated in detail by Caughlin (1994) and Zeimer et. al. (1993), and we only note its potential here as well for completeness.

6.3 Observations and Conclusions

The exercise documented in the preceding sections provided an opportunity to consider the application of abstraction techniques to an actual C^3I model. Despite the limitations that precluded a full demonstration of the abstraction techniques, there are a number of observations that we can make, and conclusions that we can draw, which we discuss them in this section. In particular, we note the following:

1. There are two different approaches to analyzing a model for abstraction opportunities; top-down and bottom up.
2. Determining which abstraction techniques to implement requires some cost-benefit analysis.
3. The original RWWM model contains examples of abstractions itself.
4. Many of the potential abstractions that we identified are interrelated.

Each of these topics is addressed in a subsection below.

6.3.1 Top Down Versus Bottom Up Analysis

There are two significantly different approaches to applying model abstraction techniques. One approach deals with the model philosophy expressed as assumptions on which the model is based. These assumptions are the basis of the applied abstractions, and any analysis of the applicability of abstraction techniques is limited to extension of or addition to these assumptions.

We may consider this approach to reflect a top down viewpoint, well suited to the classic systems design methodology.

The alternative approach to analysis of application of model abstraction techniques is from the bottom up, by examining the code itself. Analyzing such abstraction techniques has some similarities to code optimization, in that one must search through the code without a preconceived notion of the outcome, looking opportunistically for methods to simplify computation without changing model validity. Thus one could look at a specific subroutine to determine if all exogenous variables are required (i.e., model sensitivity analysis), or whether iteration rates could be decreased (i.e., temporal aggregation), or whether results could be reported in an aggregated form (i.e. function aggregation), and so on.

In this exercise, we primarily followed a bottom up approach. This was motivated by the fact that without system level design documentation, it was difficult to start from system level assumptions to perform a top down analysis. We demonstrated the viability of a bottom up approach, while recognizing that a more useful analysis can be achieved by combining this approach with a top down approach.

6.3.2 Cost Benefit Analysis

While we have emphasized the benefits of implementing model abstraction, we must also keep in mind that there is a cost associated with abstractions as well. These costs include the effort required to analyze the model, the effort required to implement the abstraction, and the effort required to validate the results.[7]

The greatest cost savings can be achieved by implementing abstractions within procedures that are frequently executed (such as LNCRLE). In looking at such procedures, we determined that they consisted primarily of straightforward trigonometric calculations with little in the way of detailed simulation that might lend itself to this type of abstraction. While some simplification could be invoked at this level (such as small angle approximations to sine/cosine relations), there would be little gain in these, especially after consideration of the cost of the abstraction logic needed.

On the other hand, we also identified abstractions that may provide little benefit in terms of computational requirements. For example, the priority computation is a sequence of IF-THEN-ELSE statements, involving up to six comparisons and a small number of arithmetic computations for each priority computation. Abstraction by reducing the number of comparisons has a negligible effect on computation. Unless there is some additional benefit to implementing this abstraction (such as reducing data collection requirements), then the benefit of this particular abstraction does not appear to be worth the cost.

[7]This entire research effort can be viewed as an attempt to reduce the cost of analyzing the model (by providing a comprehensive and organized approach to the analysis) and validating the model (by initiating steps that could result in abstraction process validation rather than individual model validation).

Thus abstractions are not all equal in terms of their benefit relative to their cost. Cost benefit analysis is therefore an integral part of the process of applying abstractions to existing models.

6.3.3 Abstractions Incorporated into Original RWWM

Several of the abstraction techniques identified in our taxonomy are represented in the original RWWM. For example, the calculations of probability of target destruction by different weapons in the subroutine CPKS is implemented as a set of look-up tables. Another abstraction involves the wingman. The wingman represents a second aircraft capable of supporting the main aircraft in its mission. However, the location of the wing aircraft is set to the location of the main aircraft. This is a state aggregation. There are a number of states in which the wing aircraft could support the main aircraft, represented by different wing aircraft locations relative to the main aircraft. These states have all been aggregated to a single state in which the wing aircraft is located exactly at the location of the main aircraft.

6.3.4 Interrelationships of Abstractions

There are a number of examples in Section 6.3 that illustrate how some abstractions are interrelated. A case in point is the launch point computation. The abstraction described in Section 6.3.2.1.1.2 replaces the original computation with a computation in which there is no aircraft altitude change, i.e., the pull up rate is zero. Since this is the only point in the model where the exogenous input PULUP is used, this state aggregation leads to a related parameter elimination abstraction. Parameter eliminations can be easily (trivially) identified when the parameter is no longer used in the model as the result of an abstraction.

Another similar example involves abstraction of the representation of aircraft maneuvering. State aggregation implemented as changes to the maneuvering that eliminate the use of parameters such as TRNRATE and TRNRAD result in parameter elimination as well.

7. Application of Abstraction Techniques to ALARM

The preceding section illustrates both the potential benefits of and the difficulties of applying abstraction techniques to an existing model. Recognizing that there is a wide variety of models in existence today, it is important to ensure that the abstraction approaches are not too narrowly focused on a particular type of model. Also, part of our objective in understanding abstraction techniques is to identify the characteristics of models that make the application of particular abstraction techniques more or less effective.

In this section, we apply the same type of analysis that we used in the preceding section to a different model. For this analysis, we selected a battlefield model of a radar system represented at a more detailed engineering level. This model, the Advanced Low Altitude Radar Model (ALARM), is currently being used for a number of analyses and studies of radar systems. In this section, we review the ALARM model with respect to the various abstraction techniques categorized in Section 5, in terms of how the technique can be applied to the ALARM model. Section 7.1 includes a brief introduction to the ALARM model and a comparison of ALARM and RWWM. The results of the analysis are contained in Section 7.2, and some conclusions and observations are provided in Section 7.3.

7.1 Summary of ALARM

We begin the technical discussion with a description of the ALARM model. This information is presented in two parts. First, a brief overview of the ALARM model is provided in Section 7.1.1. Then, in Section 7.1.2, the structure and purpose of the ALARM model is compared to the RWWM described in Section 6.

7.1.1 Model Overview

As described in the Operational Concept Document for ALARM (SAIC 1995a), the ALARM model is a computer simulation designed to determine the detectability of a single target by a single radar, intended for engineering study of radar design and detectability phenomena. The model is capable of simulating the operation of moving target indicator (MTI), pulse doppler, and continuous-wave (CW) radars. External signal power levels delivered to the radar receiver are modeled in a high degree of detail so that the user can assess not only specific systems, but the operation of those systems in a realistic environment. User supplied inputs include such engineering level data as transmit power, pulse width, pulse repetition frequency (PRF), antenna pattern, target radar cross section (RCS) tables, and data needed to simulate MTI and pulse doppler processing. A jamming module provides a means for determining burnthrough for noise jamming and signal-to-jamming ratio for coherent jamming.

Detection performance is determined over one integration period. Detections can be determined for either a sequence of points along an aircraft flight path, or a detection map in the form of detectability contours covering the entire area of interest.

Radar detection calculations are based on the signal-to-noise radar range equations commonly used in radar analysis. The model includes the environment effects of:

- atmosphere;
- terrain masking;
- clutter (including land and sea clutter reflectivity probability distributions);
- multipath; and
- electromagnetic propagation.

Pattern propagation effects such as radar antenna pattern, spherical earth and knife edge diffraction, and multipath are also included.

The ALARM detection results include the effects of MTI or pulse doppler filtering for reduction of clutter returns. Filter effects such as blind velocities (doppler) are considered, as well as other filter effects caused by the number of MTI delays and number of doppler filters and filter characteristics. The effects due to pulse eclipsing/blanking, MTI range and azimuth gating, and pulse compression are included. In addition, ALARM models onboard noise (self-screening) jammers, onboard deception (coherent) jammers, and standoff noise jammers.

7.1.2 A Comparison of RWWM and ALARM

In terms of similarities, both models include a representation of the detectability of airborne targets by ground based radar. Radar sites and aircraft are represented in each model. However, beyond the functional overlap, the differences between the models are more striking.

The most significant difference between the two models is that they represent different levels of detail in the air/ground engagement domain. RWWM is a mission level model, in which the activities include detection, weapons launch, and aircraft maneuverability. Multiple radar sites and aircraft can be represented (although combat resolution is handled as a replication of multiple one-on-one engagements). Measures of effectiveness express the outcome of the mission. ALARM is an engineering model, providing more detail in the specific area of detection, with only a single target and single radar, and the measure of effectiveness is the detectability of the target by the radar. This difference in model level has significant implications for model abstractions:

1. Since there is no sequence of events with associated states represented in ALARM, model abstractions based on time-varying state aggregation cannot be applied.
2. The relationships among model parameters in ALARM are generally expressed as mathematical equations.

3. The ALARM model is structured as a set of factors which affect the final outcome of the detection calculation.

These three implications are discussed in more detail in the following paragraphs.

Unlike RWWM, ALARM has no time clock; the factors which affect detectability are constant over the simulation time period represented in the model. Thus the concept of state is somewhat different than present in discrete event models such as RWWM. One can think of different states in ALARM, representing different combinations of input conditions; however, there is no state transition over time, as occurs in RWWM. Abstractions which are characterized by aggregating states whose transitions are irrelevant to model outcome cannot be applied to the ALARM model. Static state aggregation, the combination of system states whose differences do not impact target detectability, can still be applied. However, in ALARM, such a situation would most likely be represented as elimination of an input parameter (whose value distinguishes among different states).

The second relevant aspect of ALARM is that the model is generally expressed as a set of mathematical formulas. This attribute has the advantage that inspection of the relationship of input parameters in the model is simpler. For example, the equation used to calculate the target body signal power along the direct path from the radar to the target is:

where

S_{Tdi} = Target body signal power along the direct path from the radar to the target

C_{Ti} = Constant comprised of all constant terms used to determine the target body signal level

σ = Target body radar cross section

R = Slant range from radar to target

The relevance of slant range to target body signal power is immediately obvious. In terms of identifying abstraction opportunities, the challenge in the case of the ALARM model is the scope of the model and the factors that are present in the model. The actual detection calculation can almost be expressed as a single equation, but one with over one hundred individual parameters. At that scale, it becomes much more difficult to isolate the potential impact of a single parameter.

To manage the potential complexity of such a model, the impacts of various factors on detectability are organized in a modular manner within the model. This hierarchy is evident from the structure of the documentation contained in the OCD (SAIC 1995a). Figure 12 depicts this organization. There are three broad categories of factors which impact detectability: the strength of the signal received by the radar receiver, the processing applied to that signal, and additional factors that impact the detectability of the signal. The model is conceptually organized in this manner, and we exploit this organization in the search for abstraction opportunities.

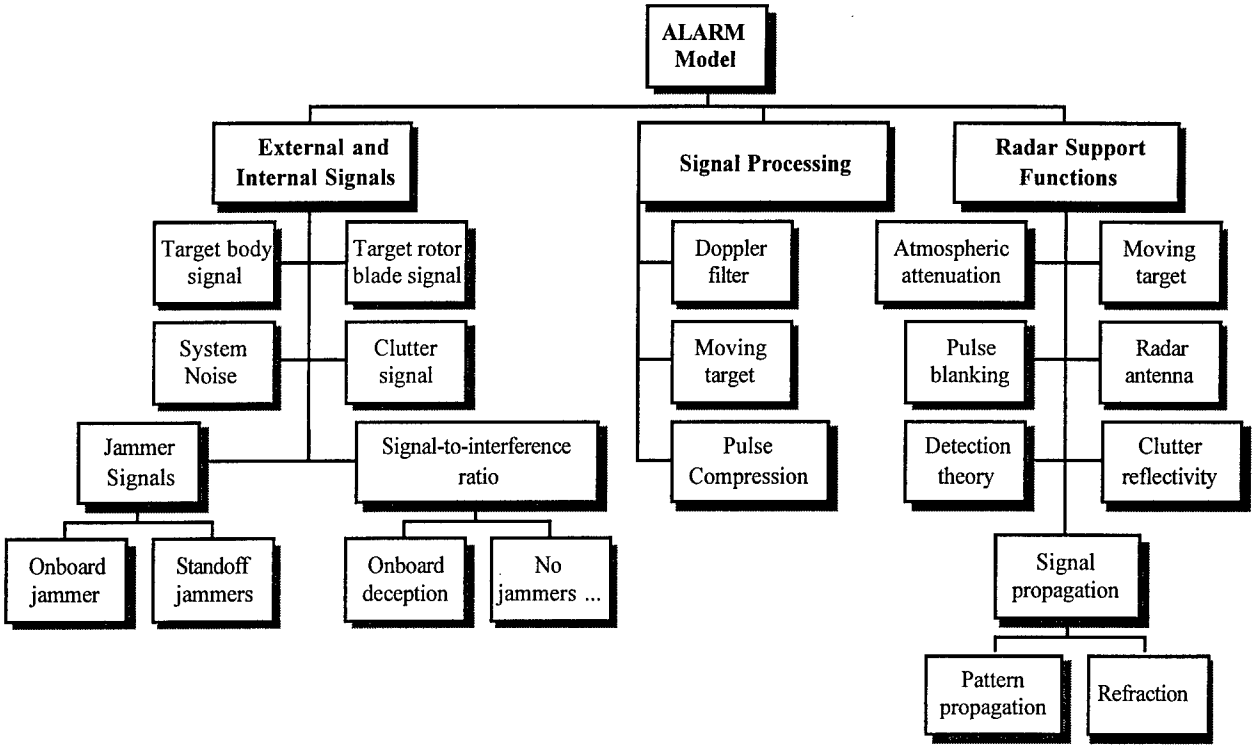


Figure 12, ALARM Hierarchical Structure

7.2 ALARM Abstraction Analysis

In considering abstractions to the ALARM model, we walk through the taxonomy of model abstraction techniques presented in Section 5 and consider the applicability of each of the abstraction techniques. Some abstraction techniques can be quickly dismissed as inappropriate for the ALARM model, while others warrant further consideration. This initial analysis of abstraction techniques for ALARM is summarized in Table 3. Discussion of specific abstraction approaches is contained in subsequent subsections.

7.2.1 Hierarchy of Models with Explicit Assumptions

The first abstraction technique to be considered is that of explicit assumptions. Each box in the modular structure of ALARM depicted in Figure 12 represents some assumption made concerning the relevant factors in determining target detectability. For example, the box labeled "Target rotor blade signal" represents the contribution of rotational movement of the target (rotor blades, engine movement) to the returned signal. The model facilitates an explicit assumption concerning the significance of a rotational component of the returned signal strength. The computational complexity of the model is reduced by explicitly assuming that there is no significant contribution to the returned signal from any rotating element of the target and thereby not invoking that element of the model. In terms of the mechanics of implementing this

assumption, the user merely does not include the appropriate parameters (the DATAROTO dataset) in the input data files.

Table 3, Initial Analysis of Abstraction Techniques Applied to ALARM

Initial Analysis of Abstraction Techniques Applied to ALARM	
Technique	Applicability
Hierarchy of Models	A form of the hierarchy of models already exists, and additional submodels could be developed, as described in Section 7.2.1
Delimit input space	There are some delimitations already built into the model, such as the allowable values for probability of false alarm ($10^{-12} \leq P_{fa} \leq 10^{-4}$); however, there are no situations of additional processing in the model which handles boundary conditions or boundary state transitions
Causal approximation	By considering the causal relationships of parameters, we can identify candidate areas for abstraction, as discussed in Section 7.2.2
Model sensitivity analysis	Individual parameters can be analyzed with respect to their individual impact on model outcome, as described in Section 7.2.3
Boundary selection by influences	Since the model generally has only a single output, the boundary does not change
Behavior aggregation	Typically defined by state transitions over time, which is not supported in the ALARM model; however, there are some possibilities for static state aggregation, as described in Section 7.2.4
Causal decomposition	A form of causal decomposition already exists, as described in Section 7.2.5; abstractions based on causal relationships over time are not applicable in ALARM
Aggregation of cycles	No cycles in ALARM
Numeric representation	Simple changes could be made, such as converting double precision parameters and computations to single precision
Temporal	Not applicable in ALARM
Function	At the level of potential functional aggregation, there is only one function in ALARM, that of target detection; thus function aggregation is not applicable
Entity by function	Since ALARM models single entities (single target, single radar), there is no opportunity to aggregate multiple entities which perform similar or identical functions
Entity by structure	Since ALARM models single entities (single target, single radar), there is no hierarchical entity structure to be aggregated
Look up tables	Some examples exist in ALARM, as discussed in Section 7.2.6
Random number generation	Could be introduced, but given the basic deterministic nature of the model, random number generation would significantly change the character of the model
Linear function interpolation	Some examples exist in ALARM, as discussed in Section 7.2.6
Metamodeling	Could be used as described in Section 7.2.7

A set of explicit assumptions can be defined based on the existing ALARM model structure that fits the concept of the graph of models introduced by Addanki, Cremonini, and Penberthy (1991). The Graph of Models concept is based on a directed graph in which the nodes of the graph are models, and the arcs are the assumptions which distinguish the models. The assumptions are defined in terms of the impact of the assumption of the final model outcome. In the case of ALARM, the assumptions can be expressed in terms of a single outcome: the detectability of the target. Assumptions can either increase or decrease that detectability. For example, the assumption that ignores the rotational component contribution to returned signal strength decreases target detectability, i.e., a model that does not include the computation of rotational component will result in a lower signal-to-interference (S/I) value, resulting in a lower probability of target detection. Conversely, including that computation in the model, e.g., retracting the assumption, leads to a higher S/I and higher probability of target detection. On the other hand, the assumption that there is no clutter increases the probability of detection, because the clutter increases the denominator of the S/I computation.

Note that in some cases the impact of the assumption cannot be determined. For example, the impact of atmospheric attenuation cannot be directly determined because it is applied to factors which appear in both the numerator and denominator of the S/I ratio (which drives the model output). The effect of atmospheric attenuation is applied to the target signal as well as the signals associated with onboard deception jammers and clutter signals. Eliminating the computation of atmospheric attenuation (independent of any other changes) may serve to increase or decrease the S/I. If the effect is comparable to both the numerator and denominator, this abstraction could result in no impact on the final outcome.

The hierarchical structure of the model depicted in Figure 12 is used as the basis for creating a candidate graph of models for the ALARM model. This graph is shown in Figure 13. The arcs are labeled with the assumptions, along with the impact of the assumption on model output and the method to implement that assumption. An assumption includes the model output variable that is affected by the assumption and the direction of change. Note that the direction of the arrow points toward the more detailed, and away from the more abstract, model. The box at the top of the picture represents the simplest model, in which most of the aspects of the model have been abstracted out. Lower level boxes represent inclusion of various factors. To illustrate, consider the arrow from the Baseline Model to the Target Rotor Blade Model. This arrow indicates that by including the dataset DATAROTO, adding in the contribution of the target rotor blade signal increases the probability that the target will be detected. Adding other factors such as jamming reduces the probability of target detection. In each case, the resulting model is the union of the baseline model with the model pointed to by the arrow.

In some cases these factors independently impact target detectability, in which case they can be combined. For example, one could define a model in the hierarchy which includes clutter and jamming, but no atmospheric attenuation. If the abstractions have the same impact on the same output variable, then the combined impact of the combined abstraction can be determined.

However, if the abstractions have opposing effects, then the effect of the combined abstraction cannot be determined a priori.

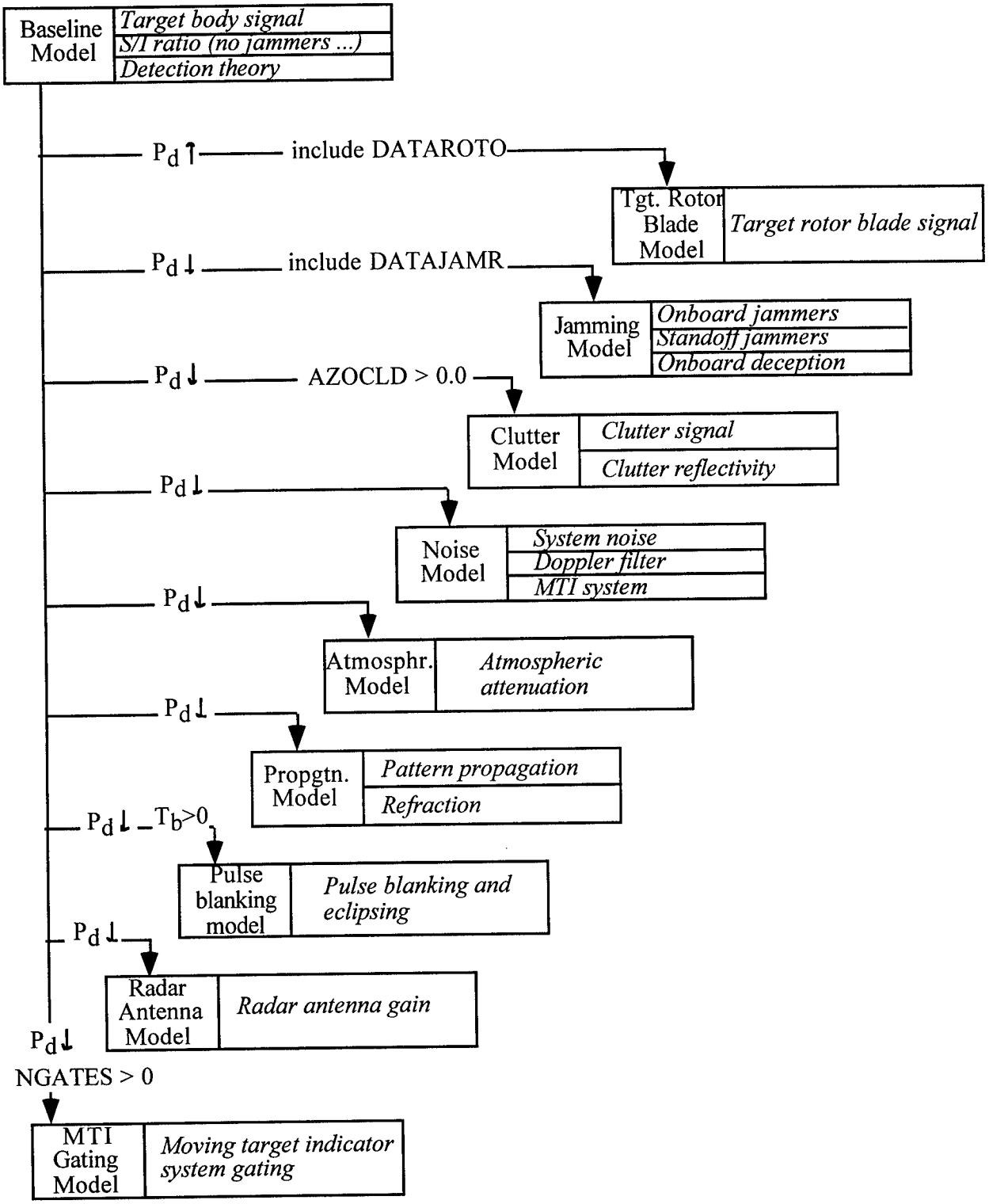


Figure 13, Candidate Graph of Models for ALARM

As shown in Figure 13, the current architecture of the ALARM model partially supports this particular graph of models. The effect of rotor blades on target signal strength is abstracted out by simply not including the data in the dataset. Similarly, the presence of jammers (for consideration in target detectability) is signified by including the DATAJAMR data set. Other abstractions can be implemented by setting specific parameters. For example, setting the pulse blanking time BLANKT (T_b) to 0.0 eliminates any pulse blanking or eclipsing, thereby eliminating this factor from the model. In the case of modeling clutter, the parameter AZOCLD is the maximum off boresight angle in azimuth for which clutter returns are computed. When set to 0.0, no clutter returns are calculated, effectively abstracting clutter as a relevant factor in target detectability.

There are several arrows in the proposed graph of models that do not have methods associated with them. This signifies that the current structure of the ALARM model has no simple way to invoke the corresponding assumption. For these assumptions, we propose development of a simplified submodel to provide an approximate result with simplified computations. The simplified computation could be as simple as a user input parameter, a table look up, or a simplified computation of the corresponding parameter.

The particular set of assumptions and decomposition of ALARM functionality shown in Figure 13 is only one of several candidate decompositions of the model in terms of simplifying assumptions. While the lowest level model (no assumptions) is identical in all cases, there are a variety of ways in which individual components of the model can be identified as the basis for assumptions.

The graph of models is used by Addanki, Cremonini, and Penberthy (1991) to navigate among the various models to find the model which provides information which is sufficiently close to the behavior of the real-world system being modeled to answer the question for which the model is being used. In the original graph of models approach, real-world observations of system behavior are compared to model outputs, and if the results vary beyond the threshold for the question being answered, an assumption is retracted which will adjust the outcome in the direction necessary to move within the threshold.

In the case of ALARM, the data would not be collected from real-world observations, but rather compared to baseline data from the full model. An initial set of test runs to determine the set of assumptions which provides sufficiently accurate results with the least computational requirements could be identified, then used for execution of experiments involving repeated executions of the model.

7.2.2 Causal Approximation

The concept of causal approximation derives from the approach defined by Nayak (1992). One method to represent the causal mapping of the parameters in the ALARM model is a parameter dependency graph (PDG). A PDG is a directed graph in which an arc from node A to B indicates that the computation of parameter B in the model is a function of (at least) parameter A. A PDG is useful in showing the processing threads associated with the various inputs to the model. The structure of the ALARM model described in Section 7.1.1 facilitates creation of a PDG, because there are no dependencies that are a function of time. Rather, for this type of model, the PDG can be derived directly from the equations which are implemented in the model.

Although time constraints preclude a formal proof that the ALARM model meets the criteria of causal mapping, we illustrate the causal ordering of model parameters with one of the ALARM PDGs in Figure 14, with parameter definitions contained in Table 4. Figure 14 depicts a high-level view of the parameter dependencies, in which some processing threads have been collapsed. The other PDGs depict similar information for the other processing threads. The full set of PDGs for ALARM are contained in Appendix D. A simplified view of causal mapping is that the parameter relationships are sufficiently modular that an abstract model fragment could be substituted for the computation of a parameter with clearly defined implications on model outcomes.

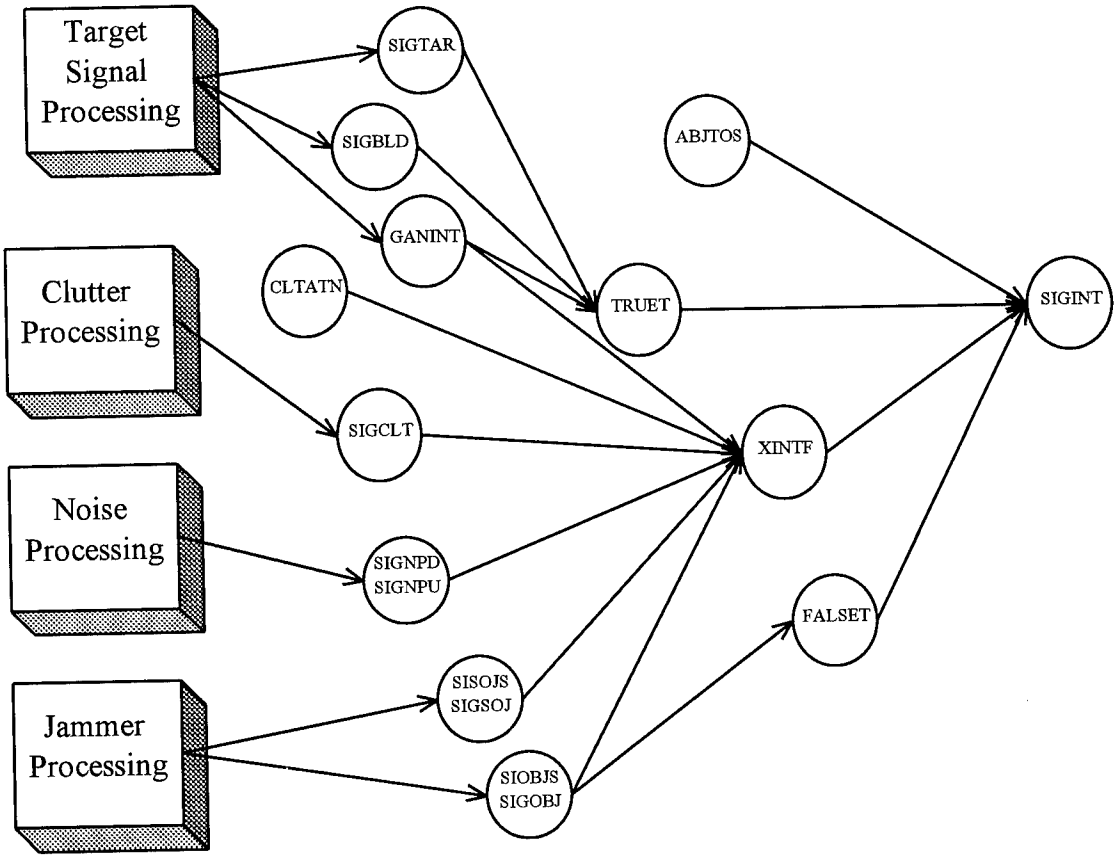


Figure 14, ALARM Top Level Parameter Dependency Graph

Table 4, ALARM Parameters

ALARM Parameters				
Parameters	Symbol	Description	Proc. Thread	Source
ABJTOS	J/S_t	Jamming-to-signal ratio necessary for the deception jamming to be effective	Top-Level	DATAJAMR
CLTATN	G_{sp}	Response of the frequency domain signal processing to the clutter signal PSD for a pulsed radar	Top-Level	Computed
FALSET	S_{FTi}	False target signal power	Top-Level	Computed
GANINT	G_I	Integration gain	Top-Level	Computed
SIGBLD	S_{Bi}	Rotating blade signal power	Top-Level Target	Computed
SIGCLT	S_{Ci}	Clutter signal power	Clutter	Computed
SIGINT	S/I_i	Signal-to-interference ratio	Top-Level	Computed
SIGNPD	S_{Ni}	System noise signal power for pulse doppler radar	Top-Level Noise	Computed
SIGNPU	S_{Ni}	System noise signal power for pulsed radar	Top-Level Noise	Computed
SIGOBJ	S_{JOi}	Onboard jammer signal power for a pulsed radar	Top-Level Jamming	Computed
SIGSOJ	S_{JSi}	Standoff jammer signal power for a pulsed radar	Top-Level Jamming	Computed
SIGTAR	S_{Ti}	Target body signal power	Top-Level Target	Computed
SIOBJS	S_{JOi}	Onboard jammer signal power for a pulse doppler radar	Top-Level Jamming	Computed
SISOJS	S_{JSi}	Standoff jammer signal power for a pulsed doppler radar	Top-Level Jamming	Computed
TRUET	S_{TTi}	True target signal power	Top-Level	Computed
XINTF	I_i	Interference signal power	Top-Level	Computed

Actual implementation of a causal approximation requires additional model fragments which provide simplified computations of key model parameters, along the same lines as required for some aspects of the hierarchy of models described in Section 7.2.1. For example, a simplified model of standoff jamming could substituted for the more detailed model currently in place, particularly if the model is being used to compare alternative radar designs (rather than predict absolute performance).

7.2.3 Model Sensitivity Analysis

Causal approximations and the hierarchy of models are both focused on abstractions which simplify a segment of the detection computation. This analysis is conducted at a "macro" level. It is also possible to conduct a more "micro" level analysis by considering the impact of

individual parameters on model outcome as well. Model sensitivity analysis can be applied to determine whether any of the model inputs are fitting parameters that can be used as the basis of an abstraction. The parameter dependency graphs provides some insight into the sensitivity of individual parameters on model outcomes. For example, parameters such as range to target (R) and radar transmit frequency (f_0) are integral to the computation of several of the components of the model output. Clearly abstraction of such parameters must be analyzed carefully because of the breadth of the impact of the parameter in the model.

Validation exercises typical for models such as ALARM can provide useful information for considering abstractions which involve parameter elimination. Often the validation process includes an assessment of the sensitivity of model outcomes to perturbations in input parameter values. Such information can provide insight into the issue of the impact on model outcome of parameter elimination.

7.2.4 State Aggregation

As noted in Section 7.1.2, time based state transitions cannot be aggregated in ALARM because there are no such transitions. However, the ALARM System User's Manual (SAIC 1995b) provides some hints for more efficient model execution that fall into the category of state aggregation, where the states are based on the decomposition of the values of state variables.

One simplification discussed in the documentation is manipulation of the size of the clutter patches used to calculate the clutter returns. A clutter map is a set of state values, in which a value represents the effective clutter characteristics of a region on the ground. Reducing the resolution of the map by increasing the size of the patches assigns the clutter characteristic to a larger area, reducing the number of states that can be represented by the map. Thus changing this resolution fits our definition of state aggregation.

Another similar simplification suggested in the documentation is the number of data points for which detectability is calculated, in either flight path mode or contour mode. The ALARM model determines a set of target coordinates, either moving along a mission flight path or covering an area in a grid pattern, and calculates detectability of the target at those points. The user inputs a parameter defining the spacing or resolution of these points (DXYPLT for contour maps, the specific data points for flight profile). Clearly the fewer points for which calculations are made, the less computational demand is made on the model. Each point for which a calculation is made is a vector in the model state space:

<radar site, target location/velocity/orientation, P_d>

Thus reduction in the number of target location points represents an aggregation of the possible model states.

These approaches are suggested in (SAIC 1995b) as means to less time-consuming execution of the model. However, the motivation suggested in the documentation is somewhat different than our approach. The User's Manual suggests that the "abstracted" (our

terminology) version of the model be run during model set up, then the fully detailed model be run to generate the final results. The essence of our abstraction approach is to reverse this process. The model set up phase includes determining the extent to which the model can be abstracted without impacting model outcome relative to the simulation requirement. This process may include execution of the model at higher fidelity first before determining the optimum level of fidelity. Of course, the benefit of our approach is only realized in situations where the final model execution involves multiple executions, and is sufficient to justify the cost associated with determining the appropriate level of state aggregation.

7.2.5 Causal Decomposition

The principle of causal decomposition is to divide a model into loosely connected components, then consider each component separately while specifically addressing component interactions. In the case of the ALARM model, the component structure depicted in Figure 12 qualifies as a decomposition, and a simple causal decomposition abstraction is already implemented in ALARM. The signal component of the S/I parameter is determined first; if the signal is less than the detection threshold, the computation of factors which reduce the S/I parameter is bypassed.

A potential enhancement to ALARM is to further this approach to determine the maximum detection range given (worst case) radar and target parameters. Then for each contour or flight path point, the range could be compared to the range threshold first, without calculating the actual signal strength. It is even possible to eliminate a set of points from consideration. For example, while stepping along a north south line in generating a contour map, stepping from a point which is within the maximum range threshold to a point beyond the maximum range threshold indicates that the rest of that line is outside the maximum range threshold, and processing can immediately move to the next north south line.

7.2.6 Look Up Tables/Linear Function Interpolation

There are several examples in the ALARM model in which look up tables and linear function interpolation are used to abstract a more computationally complex calculation. For example, the determination of atmospheric attenuation involves retrieving the appropriate value from a table, based on the elevation angle of the radar, the range to the target, and the transmission frequency of the radar. Another example is the radar antenna gain computation, in which azimuth and elevation are used to index into a table which describes the radar antenna gain pattern.

These particular functions were implemented in the form of a look up function due to the unfeasibility of expressing the function in mathematical terms. However, this concept can be extended to support an abstraction of the existing model in which the computation is expressed as a series of mathematical equations. A single table could be created as a simplified version of

the overall computation, or scenario-specific (but invariant) parameters could form the basis for creation of a table of data at program initialization.

7.2.7 Metamodeling

The final abstraction techniques discussed in this working paper is that of metamodeling. Metamodeling involves deriving an input/output relationship from sets of model input/output pairs. The advantage of a metamodel is that it can reduce a complex model into a single mathematical expression, which can then be used as an abstraction of the model, or as a tool for further analysis.

Metamodels can be developed for the ALARM model, but there are some challenges in proposing this form of abstraction. First, some analysis is required to determine what parameters are the variables in the metamodels. For example, in the TERSM metamodel study cited in Section 10, initial analysis was conducted to select aircraft altitude, speed, azimuth, and number of channels as the variables which the metamodels related to model output. ALARM has over one hundred input parameters, and similar steps are required to reduce that number to a manageable set for metamodel construction. (The specific parameters will depend on the specifics of the simulation requirement being addressed in the exercise.) Efficiency gains for metamodels must be carefully considered. Portions of the ALARM model are already defined in terms of the polynomial functions of input parameters. Those computations which are not based on polynomial functions are in many cases based on table look up. Efficiency gains in such cases may not be on the same order as those associated with metamodels which abstract results that are computed over a sequence of simulated time (as was done in the TERSM model).

Experiment- or application-specific metamodels could be developed. For example, for radar design applications, one could focus on key radar design parameters, and develop metamodels which express model results as a function of those parameters. For applications in which the key consideration is the performance of a radar against a variety of jamming threats, metamodels which express model results as a function of the jammer parameters could be developed.

7.3 Observations and Conclusions

The analysis in the preceding section is only the very beginning of the effort required to implement abstractions in ALARM. Even in the areas in which abstractions are suggested, additional analysis is required to determine the feasibility and cost/benefit tradeoff of actually implementing the abstraction. However, even in this cursory analysis we have derived some useful lessons on the more general topic of the application of abstractions to models. In this section, we present these observations and conclusions.

7.3.1 Abstraction without Modification

First, the use of abstractions in an existing model does not always require changes to the code or architecture of a model. Rather, abstractions in a well-constructed model may involve

nothing more than executing with certain data in specific circumstances. For example, we noted that one could abstract the impact of pulse blanking and eclipsing by setting the blanking time to zero in the input data file. Note that such an action does not necessarily imply that the radar being modeled has that physical characteristic; rather, it signifies that for a particular experiment, that factor is not relevant to answering the question that the simulation results are being used to answer.

We use the term **parameter nullification** to describe the situation in which there is a value to which a parameter can be set which nullifies its impact in the model. Examples in ALARM of parameter nullification include blanking time (for pulse blanking and eclipsing) and clutter azimuth (which when set to zero eliminates the computation of clutter returns). There are two important attributes of parameter nullification to consider. First, the nullifying value should result in reduced computation; if the same computation is required for the nullifying value as the other values, then there is no benefit to the abstraction. The other factor to consider is that the nullifying value must be computationally valid. For example, zero cannot be a nullifying value for a parameter which appears as a divisor in subsequent computations.

From this example, we can also extend this observation to include the characteristics of the ALARM model which support abstraction. These attributes include modularity and parameter nullification. By modularity, we refer to the separation of functionality into separate modules. The value of modularity for abstraction is that it facilitates implementing a specific assumption. If the impact of a specific physical effect is isolated in a single module, then assumptions which abstract that factor in the model can be implemented by eliminating the corresponding module.

7.3.2 Domain Expertise

A second observation is the importance of domain expertise in the selection and implementation of model abstractions. While one can identify the candidate abstractions from an analysis of model, the implications of an abstraction are generally expressed in terms of the modeling domain, particularly in terms of the simulation requirement. For example, domain expertise would be useful in identifying causal decomposition or model hierarchy abstractions by providing an understanding of the implications of simpler approximating models on the overall model results. The approximations in such cases are representations of the physical phenomena being represented in the model, and the domain understanding of such phenomena is an integral part of the abstraction process.

The necessity of domain expertise in the abstraction process is no different than the need for domain expertise in the model development. Model development is most effective when performed as a synthesis of domain expertise and software engineering skills. The implementation of model abstractions requires a similar synergy.

8. Application of Abstraction Techniques to ARES

The third model analyzed under this effort is the Advanced Regional Exploratory System (ARES) model. This model provided a different challenge in terms of defining abstraction techniques, since it is a theater-level combat simulation, at a different level of resolution than either the ALARM or RWWM models described in the previous two sections. In addition, there is a specific requirement for the ARES model to encompass variable levels of fidelity in certain aspects of the model.

In this section we present the analysis of candidate abstractions for ARES. This section is organized in the same structure as the previous two sections. We begin with a description of ARES model (Section 8.1), followed by a discussion of abstraction techniques (Section 8.2) and our observations on abstractions applied to this model (Section 8.3).

8.1 Description of the ARES Model

We begin our analysis of abstraction techniques in the context of the ARES model with a description of the model itself. This discussion is divided into three parts. Section 8.1.1 includes an overview of the model itself. In Section 8.1.2, we discuss the specific ARES requirements for variable fidelity and how abstraction techniques are essential in achieving that goal. We conclude this section with a comparison of ARES and the other models analyzed under this effort (RWWM and ALARM).

8.1.1 ARES Overview

ARES is a theater level combat simulation in a joint and combined forces context, intended for use as an analytic tool. It is being developed by merging two existing simulation models: the Concurrent Theater Level Simulation (CTLS) developed by Computer Sciences Corporation (CSC), and the Theater Exploitation Study System (TESS), developed by GRC International. CTLS provides variable resolution ground combat, maneuver networks, integrated air operations, and a command and control operations order language. From TESS, ARES draws the user interface, intelligence, deception, information flow, and rule-based command and control.

The ARES Software Design Document (SDD) identifies a number of objectives for the ARES model, summarized as follows:

- Simulate joint/combines land, sea, and maritime force operations of all services and multiple nationalities;
- Address questions of whether a given time-phased set of resources are sufficient to assure successful execution of a given regional strategy, or address the resources that are required for successful execution;
- Have the flexibility necessary to address a wide range of potential threats and operational environments;

- Be object oriented, and is capable of varying levels of resolution;
- Employ a state of the art user interface;
- Have powerful tools for file management, data development, output analysis features, and display features;
- Emphasize information flow from sensors through communications capabilities to command and control nodes; and
- Allow users to employ rulebases for simulating command and control, pre-defined scripts, or a combination of the two. (GRC and CSC 1995b)

The ARES system actually consists of four major components:

- The Human Interaction Component, which provides the graphical user interface to ARES;
- The Problem Domain Component, which models the real-world entities whose behavior is being analyzed;
- The Task Management Component, which includes the "simulation engine" an other supporting functionality; and
- The Data Management Component, which manages the data necessary to support the execution of ARES.

The aspect of ARES that is relevant to this study is the representation of the real-world system, or what is referred to as the ARES model (which is actually the Problem Domain Component). The remainder of this section focuses on the ARES Problem Domain Component.

The ARES PDC is divided into a set of modules which contain a set of objects which form a logical functionality of the model. These are referred to as "submodels" in the discussion which follows, to reflect that they are components of the overall ARES model, but contain a functionally cohesive subset of the objects and methods. The SDD provides a high level design for the following submodels:

- Air Operations;
- Ground Operations;
- Communications;
- Intelligence, Surveillance, and Reconnaissance;
- Command and Control;
- Movement; and
- Combat.

Naval operations, logistics, and mobilization are identified as submodels to be included in a subsequent development phase.

8.1.2 ARES Requirement for Variable Fidelity

Of particular interest to this study is the explicit requirement for variable levels of fidelity in the ARES model. This capability is explicitly identified in the overall statement of objectives, with specific requirements defined in the ARES Requirements document. These specific requirements are listed in Table 5. The CMPX column provides an estimate of the level of complexity in implementing the requirement (ranging from low to high), while the PRI column indicates the relative priority of the requirement. The ARES development team shows respect for the challenge of achieving multiple levels of fidelity in a model, as only one of these requirements is estimated as being of low complexity.

8.1.3 Comparison of ARES and RWWM and ALARM

There are several aspects about this exercise that differ from the analysis of models documented in the preceding two sections. In this section, we highlight those differences as a prelude to the analysis of abstraction techniques applied to ARES. These differences are as follows:

- ARES is a theater level model, in which the battlefield entities and activities are represented at a greater level of aggregation than the entities and activities in RWWM or ALARM.
- ARES includes a specifically defined requirement for variable levels of fidelity.
- For the ARES analysis, we worked with software design documentation.

Each of these issues is discussed in a paragraph which follows.

We have already explored differences in level of representation in noting differences between RWWM and ALARM. Entities and activities in ARES are represented at a substantially higher level than in RWWM, since the focus of ARES is theater level, rather than mission level. This difference in manifested in less representation of well defined physical processes, and more representation of less defined processes such as command and control. As such, ARES provides a useful foil to the other two models by presenting a different set of issues in terms of model abstraction.

Unlike RWWM and ALARM, ARES has a specific requirement for variable levels of fidelity. The search for abstraction opportunities discussed in the preceding two sections is motivated primarily by a desire to generally reduce computational requirements. In ARES, the motivation is stronger, but the job is somewhat easier in that the structure of the model is designed to explicitly support variable levels of fidelity, and by extension, abstractions.

Table 5, ARES Variable Fidelity Requirements (GRC and CSC 1995a)

ARES Variable Fidelity Requirements			
REQ. NO.	**CMPX**	**PRI**	**REQUIREMENT - ARES SHALL:**
Ground Operations			
2.1.10.1	M	1	Provide for variable levels of detail in the maneuver network
2.1.10.2	H	2	Design for variable fidelity in attrition computations
Air Operations			
2.2.10.1	H	3	Provide low-fidelity model of air operations based on mission groups
2.2.10.2	M	1	Provide medium-fidelity model of air operations based on flight groups
Command and Control			
2.5.9.1	M-H		Provide selectable rulesets
2.5.9.1.1	M	2	Provide generic rulesets to serve Red and Blue
2.5.9.1.2	H	3	Provide rulesets for different theaters and C2 styles
Communications			
2.6.5.1	H	3	Provide capability to select fidelity level on basis of side, link type, node type, or subportion of comm architecture
2.6.5.2	H	4	Provide selectable signal modeling complexity based on link delay, data rate, bandwidth, modulation/spread spectrum, encryption, and routing
Intel and Sensing (e.g., ISR)			
2.7.7.1	M	2	Provide capability to select fidelity level by sensor, sensor type, region, or side
2.7.7.2			Provide selectable fidelity features:
2.7.7.2.1	M	1	Range to target (e.g., proximity detection)
2.7.7.2.2	H	1	Target attributes
2.7.7.2.3	L	1	Day/night
2.7.7.2.4	H	3	Line-of-sight (occlusion)
2.7.7.2.5	M-H	2	ECM and CC&D effects
2.7.7.2.6	M	3	Atmospheric effects

2.7.7.3	M-H	1	Provide capability to aggregate sensors of a given type belonging to the same unit

Another difference between the analysis of ARES and the analyses of RWWM and ALARM is that the analysis is based solely on the design documentation. As discussed in the preceding section, our experience with ALARM led to some interesting observations about the relative utility of detailed (code level) versus high level (design) information in assessing abstraction opportunities. Recall that the conclusion was that both types of information, used in an iterative approach, was the best strategy for analyzing models. In the case of ARES, the design level information provided the basis for the analysis. While a number of abstractions could be identified, in some places additional design information, such as data items passed among objects, is needed to completely describe how certain abstraction techniques could be implemented to create variable levels of fidelity in the ARES model.

8.2 Analysis of Abstraction Techniques for ARES

Model abstraction techniques are an important tool in achieving the goals of multi-level fidelity. By starting with the more detailed, higher fidelity model, one can derive a multi-level capability by defining a set of abstractions of the detailed model. The abstractions ensure that there is a logical connection between high and low fidelity models, so that data can be reasonably interpreted and compared among model executions.

In this section, the application of abstraction techniques to the ARES model to support multiple levels of fidelity is discussed. Abstractions are identified at three levels:

1. Within a specific object or method;
2. Within a submodel of the model, but involving multiple objects and/or methods; and
3. Across multiple submodels.

Each of these levels is addressed in a paragraph below.

At one level, an abstraction can be identified internal to a specific object class and method. A simplified technique for executing the method could be implemented that does not change the objects or methods in the model. If the abstraction involves parameter elimination, then the interface among objects may change (because the eliminated parameter would not need to be passed), but the set of objects and methods would remain the same.

At the second level, an abstraction could change the methods and/or object classes in the model. For example, two levels of fidelity could be implemented in the air operations model by defining two `employWpns` methods for air units: one that handles single aircraft, and one for multi-aircraft sorties. When the aircraft method is invoked, the combat activities of the individual aircraft are represented separately. When using the air unit `employWpns` method, the combat

activities of the individual aircraft are aggregated to the air unit level, and therefore are represented at the lower level of fidelity.

The third level at which abstractions can be defined is across functional submodels. Such abstractions would impact methods and objects within multiple submodels, where the level of representation in one submodel impacts the functionality of another module.

While recognizing these three levels, the methodology followed in this analysis focuses primarily on the functional submodel level. The level of detail in the SDD does not support analysis of abstractions within a single method, so the first level of abstractions is not addressed in this study, although this level should be revisited as part of a detailed design phase. And while the level of detail in the SDD supported the third level (across functional submodels), such abstractions were generally motivated by an abstraction in one functional submodel that influenced methods in other functional submodels. Thus such abstractions were identified by analyzing individual submodels without an explicit phase looking at combinations of submodels. In the discussion which follows, some examples of such situations are cited, but since the motivation can be traced to a single submodel, these abstractions are discussed in the context of that submodel. Thus the primary focus of the methodology is the second level of abstractions cited in the preceding paragraph, and this section is organized in terms of ARES functional submodels.

For each submodel, the types of abstractions that can be applied are considered. In some cases, the ARES design specifies functional capabilities at varying levels of resolution. This structure provides a useful illustration of the value of applying abstraction techniques to existing multi-model structures. The specific model design criteria are mapped to techniques defined in the taxonomy of Section 5. By identifying the techniques, the soundness of the abstraction can be evaluated. If the abstraction is not sound, knowledge of abstractions can lead to some design modification which results in sound abstractions.

Note a distinction between use of certain terms in the ARES documentation and the use of the terms in the Model Abstraction Techniques effort. In the ARES design documentation, reference is made to object states and state transition diagrams. This notion of state is much broader than the state definition used in our discussion of model abstractions. The ARES notion of state is a very broad term which denotes a situation or context which typically defines the behavior of the object. The definition in this study is narrower, in that a state is the composite of the values of all parameters of the object, which could include the ARES contextual state as one of the parameters. This distinction in interpretation of the term "state" is particularly important in considering state aggregation abstractions in ARES. Such abstractions are based on states as defined herein. Typically multiple such states are brought together within one ARES object state.

In this section, each functional submodel is analyzed to identify opportunities to implement simpler versions of submodel components. In most cases, the abstraction is identified

as an alternative method for a particular object, or an alternative set of parameters to be passed from one object to another. Since the ARES Software Design Document (SDD) describes the functionality of the methods in general terms, this study is limited in many cases to general observations and recommendations for abstraction approaches. Almost all of the techniques described in Section 5 could be applied to each of the submodels, although the details at this point cannot be specified. In the discussion which follows, emphasis is placed on the abstraction techniques which are best suited to the submodels, and some examples are provided to illustrate the potential abstraction. In some circumstances the SDD explicitly identifies functionality to be implemented as variable fidelity. In these cases, the simpler versions of methods are treated as the result of abstractions of the more detailed versions of the methods, and these abstractions are analyzed to determine if they are sound abstractions. This section is organized based on the overall ARES model structure, with a subsection for each of the major submodels of the ARES model.

8.2.1 Air Submodel

The ARES Air submodel models air warfare in a joint campaign. This submodel is based on a number of objects specific to air operations, including:

- Air Unit
- Airbase
  - Ü Runway
  - Ü Shelter
- Aircraft
- Air Directive
  - Ü Guidance
  - Ü Air Apport
- JIPTL
  - Ü Target
- Air OPORD
- Air Tasking Order
  - Ü Air Alloc
- Tasking Order
  - Ü Destination
- AIRSUPREQ
- ALLOREQ
- AIRSITREP
- AIROPREP
- MISREP

Air units can move, employ weapons, employ sensors, employ countermeasures, consume ammunition and fuel, be loaded and unloaded, be attacked and suffer losses, and receive supplies. The methods associated with the air unit comprise most of the functionality of the Air

submodel.[8] Activities of air units are directed by messages and reports that are created by the Air unit command posts (CPs).

Methods associated with air objects can be categorized at three levels:

1. Trivial "get" and "set" methods which merely retrieve or store data.
2. Simple operations, such as `fuelAircraft`, `rearmAircraft`, `repairAircraft`, and `repairRunway`. These methods are characterized by similar sequences of functions: determine what is needed, schedule necessary equipment and personnel resources (if necessary), implement the action, and release the personnel and equipment (if necessary). Aircraft repair includes the additional functions of handling repair parts from inventory.
3. Complex planning functions, implemented as methods associated with Air CPs.

The first category of methods is too trivial to realistically abstract, since they involve simple storage and retrieval of data.[9] The other two categories are each discussed in a paragraph which follows.

The second category provides more opportunities for abstraction, although the `fuelAircraft` and `rearmAircraft` methods are still simple. In each case a request is received, the availability of the resource is verified, the operation is conducted, and the information about the resources is updated. There appears to be little opportunity to reduce the fidelity of this aspect of the model, other than aggregating multiple types of fuel or ammunition, or some simple function aggregation. One such possible abstraction approach is to aggregate functions of ground and air-to-air refueling. Under the Bomber Penetration model development effort sponsored by Rome Laboratory, such functions were aggregated by treating an airborne tanker as a ground site at altitude, rather than a separate entity with a separate function.

The `repairAircraft` and `repairRunway` methods are more complex in that they involve the scheduling of resources required to implement the repairs. This scheduling could provide an opportunity for abstraction, based on the complexity of, and the factors associated with, the scheduling decision. Parameter elimination abstractions could be implemented to reduce the number of factors used in scheduling resources. Types of resources could be aggregated to simplify the decision algorithm as well.

The third category of functions in the Air submodel actually involves methods of CPs, whose generic activities are described in the discussion of the C2 submodel (Section 8.2.5). However, some of the data flows defined for specific CPs provide clues as to where we can focus

[8]There are some methods associated with Aircraft, but they deal specifically with fuel and ammunition resources. Some of the methods for an air unit, such as repair, are applied to the individual component aircraft of the air unit.

[9]Recall that we bounded abstraction techniques to those techniques which change the conceptual model. Thus while there may be modifications of the data structures to accommodate more rapid database access, such modifications are considered code optimization and fall outside our definition of model abstraction techniques.

attention in this analysis of abstraction opportunities. In terms of entity attributes, elimination of Aircraft attributes such as `WeatherLimits` and `DayNightLimits` would remove constraints on aircraft utilization. This abstraction should simplify the planning process, while increasing model MOEs related to aircraft performance (such as number of targets destroyed). Another example of parameter elimination abstraction is based on data flows. The SDD depicts the data flow for the Tactical Air Control Center (TACC) C2 plan method. Inputs to this method include guidance, apportionment, targeting, air corridors, subordinate status, air defense corridors, and targets. One possible variable level fidelity capability can be created to simplify this planning model, particularly if it is possible to abstract the Advanced Tasking Order (ATO) to create a simplified planning method. While it is conceivable to totally eliminate one (or more) of the parameters listed above (such as eliminating the air defense corridors), such an abstraction would probably reduce the quality of the ATO beyond a useful point. However, it may be possible to eliminate some attributes within these data items that are relatively insignificant to the ATO generation process. These examples are all based on explicit assumptions in which an a priori decision is made as to which parameter(s) can be eliminated. Sensitivity analysis could also be applied to the implemented ARES baseline model to identify parameters whose input is relatively insignificant to the air operations outcome.

It is also at this level that the requirement for variable fidelity can be met through entity aggregation. Requirements 2.2.10.1 and 2.2.10.2 define requirements for two levels of model fidelity, based on aircraft flight groups (higher fidelity) and mission groups (lower fidelity). The key to entity aggregation in this context is maintaining a mapping between entities at the two levels; that is, there should be a mapping of flights to missions, with corresponding linkages between the various attributes of the entities. For example, the target list for a mission group should be the aggregate of the target lists of the individual flights. Likewise, there must be a correspondence between associated methods in related submodels. For example, there may be two different attrition calculation methods for attrition of air units from ground-air combat, with the results of the lower fidelity model derivable from the higher fidelity model.

8.2.2 Ground Submodel

The Ground submodel models ground warfare within the context of a joint campaign. Ground unit objects can be organized in a command hierarchy, with specific instantiations for specific units (e.g., U.S. VII Corps). Command structures are flexible, and the appropriate level of activity can be represented based on the requirements of the scenario. There are several subclasses of ground units to represent functional entities of a ground force, including engineering, fire support artillery, air defense artillery, army aviation, maneuver, special operations forces, combat systems support, electronic warfare, and intelligence. The Ground submodel also includes installation and equipment objects. There are a number of types of installations, such as ports, command facilities, air bases, telecommunications facilities, supply depots, petroleum, oils, and lubricants (POL) production facilities, repair depots, nuclear, biological, and chemical (NBC) production facilities, surface-to-surface missile facilities, military equipment production

facilities, and power production facilities. There is also a variety of messages and orders that can be transmitted from the CP associated with one unit to the CP of another unit.

There are different approaches that can be investigated to provide such a capability in the ARES Ground submodel. In this section, two approaches are addressed in detail.

- Hierarchy of models with explicit assumptions; and
- Abstraction by entity aggregation.

Each of these approaches is discussed in a subsection which follows. The final subsection introduces some additional techniques that have potential application.

8.2.2.1 Hierarchy of Models

One approach to abstracting the Ground submodel is to consider building a hierarchy of functionally equivalent objects differentiated by explicit assumptions concerning the factors used in modeling unit activity. For example, the highest fidelity version of the `employJmrs` method for the class `EWUnit` could determine optimum location, frequency coverage, and transmission strategy. Simpler versions of the same method can be created by explicitly making some assumptions about the unit's activity. One assumption could be that explicit frequency coverage is not important, eliminating that strategization from the EW unit model. In this particular case, such an abstraction could involve multiple functional submodels if the abstracted method impacted the functionality represented in the intelligence, surveillance, and reconnaissance (ISR) submodel.

This type of abstraction could be applied to almost of all of the methods that involve some non-trivial processing. Non-trivial means methods which involve more computation than storing or retrieving data. For example, for the class `ADAUnit`, the method `checkAmmo` (to return the amount of remaining ammunition supplied) is considered trivial since it involves a simple retrieval of data. On the other hand, the method `employWpns` (to employ weapons in support of a fire support mission) is considered non-trivial because it involves more than a simple retrieve or store operation, and could be abstracted by explicit assumptions.

8.2.2.2 Entity Aggregation

The other approach discussed in detail in this section is abstraction by entity aggregation. Military hierarchies are a natural area in which to apply this abstraction, because the hierarchical structure is already defined. The specific ARES ground unit functions are defined at a sufficiently high level that they do not inherently represent any particular unit level. For example, the Ground Unit Functions shown in Figure 8.24 of the ARES SDD can be divided into two groups. The first group of functions reports status and attacks to the units' CP. The second group involves actions taken in response to the receipt of a course of action (COA) from the CP, and includes movement and executing specific subclass instructions. The associated sensors and weapons, and the OPORDs on which the COAs are based, can be tailored to the level unit represented by the particular entity.

There are functions which vary at the more detailed level depending on what type of units are being modeled. Appropriate temporal and geophysical scales vary with the level of a unit in the hierarchy. A small rise that has tactical significance to a platoon is ignored at the division level. Likewise, minute by minute updates of a division are grossly inefficient, while 12 hour updates of platoons are discontinuous. Thus a variable fidelity capability must be able to accommodate corresponding time and spatial scales, but with a sound abstraction from the model of lower level units to the model of aggregated, higher level activity. Meeting this requirement is much easier if the mechanisms of unit activity are identical regardless of unit level. One approach to accommodating these seemingly contradictory requirements is use common methods for unit activities, but use level-specific plans for each level unit.

One example of a common method for unit activity is a reference point based unit representation for locating and moving units. This approach uses a single location, called the reference point, as the basis for unit location and movement in the model. Subordinate unit locations are defined in terms of the relative location of their reference point to the parent unit reference point based on the tactical posture of the parent unit. For example, a unit in road march could have a reference point defined as 5 miles behind the parent unit reference point along the route of march. Movement for a unit could be determined based on an appropriate time scale for the unit level, and would be implemented by computing the new location for the unit reference point.

If movement is unconstrained, the lower level model abstracts perfectly to the higher level model (meaning that the results of modeling the individual activities of the lower level entities would result in the same model output as that of modeling the higher level entities). However, movement is constrained by vehicle capabilities, terrain, opposing forces, and so on. Thus the model of higher level entities is an approximation of the model with lower level entities, as illustrated by the following example. Figure 15 illustrates a unit comprised of four subunits traveling along a path. The subunits' locations are based on a position relative to the reference point of the parent unit (shown as a darker circle), expressed in a Cartesian coordinate system oriented in the direction of movement of the parent unit reference point. In this modeling paradigm, only the parent unit reference point need be updated, but the locations and velocities of the subordinate units can be retrieved at any point if necessary. Note, however, that the subordinate units' path is only an approximate model. In the example shown in Figure 15, unless the parent unit slows down, the unit farthest from the reference point must travel at a much greater velocity to cover a greater distance around the curve. Such a model abstracts certain constraints in coordinated unit behavior.

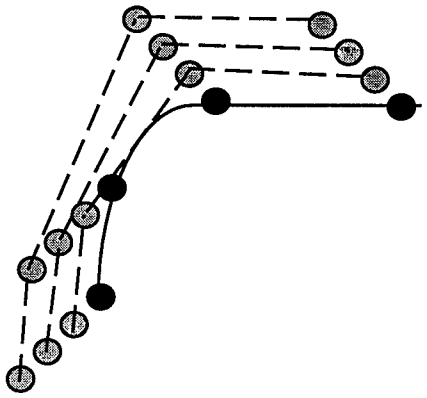


Figure 15, Movement Based on Relative to Reference Point Approach

The advantage of the reference point approach is that spatial relationships are hierarchical in such a way as to be independent of the specific level of the unit. Alternatively, more specific abstractions could be implemented that are specific to the particular level of the units involved. Unit location could be represented differently for each level. For example, unit locations of individual vehicles could be represented as individual points, locations of a squad of vehicles as the endpoints of a line, and locations of regiment as rectangles or as polygons with an arbitrary number of sides. While this approach is attractive because it allows methods for units to be suited toward the level of the unit, it has the disadvantage of requiring a different mapping among each pair of unit levels. The abstraction from company to brigade would not be the same as the abstraction from brigade to division.

There are other aspects of entity abstraction that must be considered as well as movement. Movement modeling by definition involves changes to the spatial relationships among entities. Combat involves changes to the relationship of combat power and capability among entities. In the reference point example for movement, each spatial relationship of an entity is defined in terms of its relative relationship with its parent entity. For combat capabilities, this relative relationship could be expressed as a fraction of the overall parent unit strength. Then in the abstract model, combat losses are assigned proportionally to the component units. When executing the model at the detailed level, combat losses not only alter the absolute combat strength of the unit, but also the relative apportionment of combat strength within the parent unit.

The relative relationship model does not extend well into the C2 aspects of the model. For movement and combat, the approach is based on the fact that quantifiable relationships exist between the higher and lower level units in the hierarchy. The output of the C2 submodel, the decisions made for a unit, does not map to a quantifiable value, and is therefore more difficult to express in relative terms. C2 decisions which dictate movement or combat results are level-specific and do not include easily quantifiable relationships that exist for each level to level transition.

8.2.2.3 Other Potential Abstraction Approaches

There are several other abstraction approaches that might be applicable to the Ground model, but are based on a detailed design or actual model code, and can only be introduced at the conceptual level.

One such category of abstractions involves parameter elimination based on model sensitivity analysis. The level of detail in the input to the objects can be varied by eliminating parameters in the inputs. Model sensitivity analysis is conducted by analyzing the model for parameters which are not relevant to the final outcome of a particular experiment.

Multiple levels of fidelity can be designed into a model using abstraction by parameter elimination using the concept of parameter nullification. As described in Section 7.3.1, we use the term **parameter nullification** to describe the situation in which there is a value to which a parameter can be set which nullifies its impact in the model. For example, to nullify the effect of supply consumption, the rate of consumption could be set to zero. Then without changing code, no battlefield activity would ever reduce the level of unit supplies, and the effect of loss of supply and resupply would be abstracted.

Another abstraction technique that can be considered once the baseline model code or detailed design is available is that of state aggregation. Certain model states can be combined when their distinction is not significantly relevant to the final outcome of the model execution.

8.2.3 Communications Submodel

The Communications submodel is an example of variable level fidelity model structure in which abstractions can be applied to an existing multi-level model to determine the consistency of the model. The ARES documentation points out that the user can select different levels of fidelity based on the requirements of the simulation, and includes a specification of several levels of functionality to be represented in the model.

The issue is then whether the results are consistent across multiple levels of fidelity. The concepts of model abstraction can be used to address this critical question. Specifically, if a lower fidelity model of the same functionality is a sound abstraction of the higher fidelity model, then one can conclude that the results of the lower fidelity model will be the same as those for the higher fidelity model for a variety of simulation requirements. If the lower fidelity model is not a sound abstraction, then such a conclusion cannot be drawn. Clearly the simulation requirement is a function of whatever experiment is being conducted using the ARES model, which cannot be judged by merely reviewing the model. Thus the key question is whether the lower fidelity models generate bounded, predictable variances from the more detailed model.

The first step in considering whether the abstractions are sound is to determine the impact of the abstraction on model outcomes. Typically measures of effectiveness (MOEs) are considered, but here there is a submodel which is part of a larger model. Further, the communication submodel does not generate a specific parameter that can be considered the

submodel output. Thus the analysis is based on the submodel outcome as the measure of the submodel impact on the larger model. This impact can be expressed in terms of two fundamental parameters: whether the message was sent, and the time required to transmit the message from the sending node to the receiving node. Note that the first of these parameters is a Boolean value. Predictability of an abstraction for a Boolean variable means that the output parameter can only change in one direction as a result of the abstraction. For example, an abstraction is predictable if one of the Boolean results of a higher fidelity model is guaranteed to be the result for the lower fidelity model. In other words, if an abstraction can change the outcome from true to false, it cannot also change some false outcomes to true. Some specific examples are cited below.

First the six levels of fidelity in the communications submodel are summarized, as defined in the ARES Software Design Document (GRC and CSC 1995b):

- Level 0: Single perfect communication links.
- Level 0+: Multiple communication links where each link has corresponding link characteristics such as encryption level or media type.
- Level 1: Allows multiple disjoint networks within the communications infrastructure.
- Level 2: Allows link-specific data rates connecting the nodes within the communications infrastructure.
- Level 3: Includes the effect of jamming and propagation.
- Level 4: Includes capability for dynamic routing of the message.

The analysis follows the abstractions from each level to the level above it, starting with Level 4 and working to Level 0.

The abstraction from Level 4 to Level 3 involves eliminating the modeling of a dynamic routing capability in the network. This abstraction is best described as a function abstraction. The function of the network in terms of relaying a message from source point to destination point is represented in both the baseline (Level 4) and abstracted (Level 3) versions of the Communications model. The difference is that the route determination function is performed at each node in the baseline version, and only once in the abstracted version. The advantage of dynamic routing is that it can accommodate changes to the network while the message is in transit. Such changes can range from topological (where an intermediate node or link is rendered inoperable) to traffic loading (a particular link is overwhelmed). Under most circumstances the dynamic route will be identical to, or better than, the route selected when the message is initially sent, where "better" is defined in terms of the MOEs of probability of message arrival at destination and time required to transmit message. However, in some cases an initial promising route may, due to some real-time variations to the network, end up being a poorer choice than the original selection. Such a model might actually more accurately represent the distribution of

message transmission times, but it presents an interesting challenge in determining whether the results of the abstraction are bounded or predictable. For an individual message, the abstraction is not completely predictable, in that while in most cases the abstracted model will model end-to-end transmission times as longer than the detailed model, there are certain pathological cases which can result in the abstracted model computing a shorter time. Further, it is possible (even though highly unlikely) that a series of link disruptions could result in arbitrarily long transmission times, such that the difference between higher and lower fidelity models is unbounded (for a single message transmission). However, if we look at this problem from a different perspective, we can establish the case for the soundness of the abstraction. First, if we consider the mean transmission time of the two models, the mean of transmission times in the detailed model will be less than the mean for the abstracted model. Thus while predictability is not guaranteed for individual messages, the overall effect on a large number of messages can be predicted. Further, intuitively the distribution of transmission times in the detailed model will be closer to the actual distribution of transmission times than the distribution generated by the abstracted model.

The difference between the Level 3 and Level 2 Communications submodels is the elimination of jamming and propagation effects on message transmission performance. This abstraction is a model boundary abstraction, and is best described as an explicit assumption. Without jamming and propagation effects, the communication links can function at full capacity, which provides the upper bound on the output parameters of the Communications submodel. The effect of this particular assumption is bounded and predictable, so we consider it a sound abstraction.

The next abstraction, which is reflected in Level 1, is the elimination of link-specific data rates and certain transmission priorities that can affect the route selection through the communication network. The data rate associated with a link is an exogenous input parameter that is eliminated by this model boundary abstraction. The practical effect of the rate information is that it constrains the choice of route by adding in another factor to consider. This constraint results in some candidate routes (that otherwise represent an optimal solution) being eliminated from consideration. Thus the abstraction of eliminating the constraint yields optimized transmission paths and rates; here again the abstraction is bounded and predictable.

The abstraction from Level 1 to Level 0+ eliminates the presence of multiple disjoint networks within the communications infrastructure. By eliminating the multiplicity of networks, this abstraction takes a set of separate networks and aggregates them into a single network. Clearly this abstraction is an entity abstraction. Instead of selecting a specific type of network for a particular message type, an aggregated network is used.

The final abstraction, from Level 0+ to Level 0, eliminates the physical characteristics of the transmission media. Links are represented as perfect connections between the source and destination points. This abstraction aggregates the entire network into a single perfect point-to-

point connection. As such, it can be considered an entity aggregation, with a predictable and bounded result.

8.2.4 Intelligence, Surveillance, and Reconnaissance Submodel

The Intelligence, Surveillance, and Reconnaissance (ISR) submodel contains the functions of sensor employment and intelligence collection and dissemination. Like the Communications submodel, the ISR model design specifies multiple levels of fidelity in the functionality to be implemented for the ISR entities. The discussion of the ISR model begins by looking at the variable levels of fidelity specified in the ARES design. Table 6, taken directly from the SDD, summarizes the three levels of fidelity. The abstraction represented by moving from the High fidelity version of the model to the Medium fidelity version is addressed first, followed by a discussion of the abstraction from Medium to Low fidelity versions of the model.

8.2.4.1 High-to-Medium Abstraction

In considering the relationship between the High and Medium fidelity versions of the model, each of the six functional areas is addressed, followed by a discussion of the overall model.

In terms of sensor modeling, the difference between the high and medium fidelity models is a simple entity aggregation. The individual sensors represented in the high fidelity model are aggregated into single entities (for each sensor type). Since the aggregation is only by sensor type, the functions being performed by the individual sensors in the high fidelity model are identical. The design does not indicate details of how the aggregation is performed. Some sample approaches are postulated and analyzed below.

Table 6, ISR Submodel Variable Levels of Fidelity (GRC and CSC 1995b)

ISR Submodel Variable Levels of Fidelity						
Fidelity Level	**Sensor Modeling**	**Tasking**	**Coverage**	**P_d**	**Target ID**	**Reporting**
LOW	Single Super Sensor collocated with main command post (CP)	Intel cycle; single PIR (collect everything in area of interest)	Entire area of interest (AOI)	1.0-	Collect all target attributes	Single sensor report; perception updated with latest data
MEDIUM	Single aggregated sensor for each sensor type	AOI; intel cycle; applicability mask; maximize coverage of AOI	Fraction of AOI; no explicit sensor movement	1.0 or table lookup; day/night, weather, target reflectivity	Target ID mask for each sensor type; measurement errors	Aggregated sensor reports; Spot Reports to Fire Support; perception updated with fused data
HIGH	Individual sensors	Requirements modeled; detailed resource selection and tasking	Computed using sensor motion and viewing constraints	Table lookup; day/night, weather, terrain, foliage, countermeasures	Target ID mask for each sensor type; measurement errors	All-source production; perception updated with weighted data; Spot Reports to Fire Support; distributed INTREPs

- The locations and areas of coverage of the individual sensors could be maintained, but targets within the union of the area of coverage are only processed once (rather than multiple times if within the areas of coverage of multiple sensors). This approach results in fewer target detections since a target has only one opportunity to be detected, and the aggregated sensor may not have the optimum capability to detect the target.

- The aggregation could also be implemented by using only a single sensor's area of coverage, but increasing the probability of detection as a function of the number of sensors being aggregated. Here the impact on output is less clear without an extensive mathematical analysis of the abstraction approach. One simple approach using a probability of detection scheme is to determine the probability of detection by one sensor (P_d), then compute the probability of detection if the target is visible to n sensors as $(1\text{-}P_d)^n$. However, whether the result of the abstraction results in greater or fewer target detections cannot be determined analytically. Also, this approach introduces some randomness into the model output, which may not be appropriate for ARES.

- The aggregation could also be implemented by increasing the capabilities of the aggregated sensor. The enhanced capabilities could be derived using a metamodel or table look up based on multiple executions of the higher fidelity model with

multiple sensors. The challenge in this approach is to determine whether there are parameters in addition to the number of sensors (such as the extent of overlap in areas of coverage) that impact the submodel outputs (such as detected targets). This abstraction approach is actually an abstraction by modification of model form, even though the motivation is to aggregate entities.

The preceding example shows that there are multiple ways to implement an abstraction. In this case, several implementation approaches to an entity aggregation by function are provided.

The abstraction in the tasking aspect of the ISR submodel is based on reducing the criteria used to assign collection resources. Only one priority intel requirement (PIR) is considered in each intel cycle, eliminating the processing required in the detailed model to map multiple PIRs into individual specific intel requirements (SIRs). This function aggregation is actually driven by a submodel boundary abstraction, the elimination of source, time-criticality, and intel collection priority from the tasking determination. Elimination of source is consistent with the related abstraction of sensors described in the preceding paragraph. The impact of this abstraction on model output is difficult to assess, since the model outputs are not directly measurable. The effect of the abstraction is realized in the quality of the ISR resource allocation. If the effect of the other abstractions is to effectively increase the availability of resources (by aggregating resources into a more capable set), then the impact of the resource allocation is less significant. On the other hand, if the resource aggregation results in comparable resource capability, but with less flexibility in the allocation, then the ultimate result should be fewer detected targets. However, this relationship is impossible to establish mathematically, and would require some specific simulation runs to verify.

Another dimension of the abstraction of the high fidelity ISR model is in the area of sensor coverage. This abstraction also involves reducing the set of parameters input to the model (used to determine the area of coverage of the sensor). In this case, since the sensors are represented and tasked at an aggregate level, parameters reflecting individual sensor viewing constraints (such as range, elevation, azimuth, swath width, and so on) are eliminated. This model boundary abstraction is caused by the entity (sensor) aggregation. Eliminating these parameters has predictable results, since removing the constraint which limits effectiveness results in more effective sensors (i.e., greater numbers of target detections).

The detectability of the target in the high fidelity model is a function of weather, illumination, and terrain obscuration (to reduce the number of potentially detectable targets), and a table look up for the collector/target pair.[10] The appropriate abstraction to the medium fidelity model is elimination of those parameters that are dependent on the characteristics of individual sensors. In this case, weather and illumination are treated as uniform throughout the AOI, but terrain is based on the line of sight from a specific sensor platform to target (and is clearly not

[10]The Software Design Document points out that the table look-up is used "as opposed to link analysis and thresholding." Even in the high fidelity model, there are clear examples of the use of abstractions.

uniform throughout the AOI). Thus terrain is eliminated as a parameter in the target detection calculation. Note that the effect of this abstraction is predictable, since by eliminating terrain as a factor, a target has a higher probability of detection in the medium fidelity model than in the high fidelity model. An alternative abstraction is to use an aggregated measure of the terrain variability in the AOI to approximate the target probability distribution (in much the same way that a metric for the illumination conditions in the area of interest is used). In this case, the abstraction is an entity (terrain) aggregation rather than a parameter elimination. The advantage of this alternative is that it provides a means of including the effect of terrain in the medium fidelity model, rather than essentially treating all areas as flat terrain. The disadvantage of this alternative is that some model execution is required to generate useful approximations of aggregated terrain effects.

The final aspect of the abstraction from the high to the medium fidelity model is in the reporting of sensor data. Since the data is collected from an aggregated sensor, the report information is also aggregated (an example of state aggregation). The aggregation also results in some information which is not output by the abstracted model, such as intelligence reports (INTREPs).

8.2.4.2 Medium-to-Low Abstraction

The second set of abstractions that we consider are those which create the low fidelity version of the ISR submodel. This discussion assumes the medium fidelity submodel as the baseline, as described in Table 6.

The aggregated sensors for each sensor type found in the medium fidelity model are further aggregated into a single multi-disciplined sensor in the low fidelity model. This abstraction is also an entity aggregation.

The implication of this abstraction is reflected in the abstraction of the tasking calculation. Since there is only a single sensor, no tasking criteria are required for allocating coverage areas to individual sensors. The tasking becomes somewhat trivial, in that the tasking is to collect all information within an AOI. This abstraction is a function aggregation driven by a parameter elimination.

Coverage in the low fidelity model is simply the AOI of the unit; locations of sensors are not incorporated into the calculation because no individual sensors are modeled. Since there is clearly data not input to the coverage determination, this abstraction is an example of parameter elimination.

In terms of determining the detectability of the target, the abstraction to the low fidelity model eliminates the entire computation, since any target in the AOI is considered detected. Clearly the effect of this abstraction is to overestimate the number of targets detected (relative to the higher fidelity models).

The abstraction to the low fidelity model also affects the target identification computation of the ISR submodel. Target access constraints of day/night and weather are eliminated (abstraction by parameter elimination). Also, measurement errors are not determined (abstraction by state aggregation). The rationale for considering this as state aggregation is based on considering the perceived or detected location of a target as a state. When a target is detected by a sensor, there are different possible states representing detected location, based on measurement error. All those states are aggregated to a single state in which the detected location of the target is the single actual location of the target (because the measurement error is zero).

The major difference in the low fidelity model in the area of reporting is that the information is provided in a single sensor report (since there is only a single sensor represented), and the reliability and credibility of the report is defined as perfect data. These abstractions are state aggregation abstractions, based on the same rationale as described in the preceding paragraph.

8.2.5 C2 Submodel

The C2 submodel simulates the intelligent behavior of the command posts (CPs) in ARES. The intelligent behavior is represented by a combination of algorithms and knowledge-based rules (KBRs). The algorithms are used in situations in which the intelligent behavior is based on assessments that rely on, and generate, quantitative information. The KBRs are used when qualitative measures are the basis of the decision-making process.

The Requirements Document includes a requirement for selectable rule sets, with a generic base of rules augmented by theater specific rules (requirements 2.5.9.2, 2.5.9.1.1, and 2.5.9.1.2 in Table 5). The transition between generic and specific rule sets is a special type of abstraction. The abstraction is from the specific (more detailed) to the generic (less detailed), and is referred to as generalization. A generalization abstraction provides the required multiple levels of fidelity. Generalizations can be implemented using a variety of abstraction techniques. For example, elimination of a parameter which distinguishes the specific situation from the generic situation is a generalization abstraction. State aggregation can also be used to implement generalization if the states being aggregated differ only in state values which distinguish specific aspects of the same generic situation.

Beyond the requirement for selectable rulesets, there are no comprehensive concepts for variable fidelity versions specified in the SDD for the C2 submodel. However, there are two examples cited in the SDD in which lower fidelity versions of the baseline methods are suggested. In this section these ideas are analyzed first, before discussing additional concepts for variable fidelity options in the C2 submodel.

The first suggestion for variable fidelity involves the elapsed time for a C2 decision to be made. The highest fidelity model includes a computation of the delay time as a function of the availability of personnel and computing resources. This approach can be abstracted to a fixed elapsed time for decisions of a particular type by eliminating parameters such as the resource

requirements to make the decision and the availability of those resources. This functionality can be further abstracted by using a fixed elapsed time for all decisions. This abstraction is a further parameter elimination abstraction, in which the type of decision is eliminated as a factor in the modeling of the C2 decision time lapse. Consistent with the "softness" of the C2 submodel, the actual impact on model outcome of this abstraction is difficult to assess. Delays in making decisions should have negative consequences in terms of the operational effectiveness of the entities being modeled, but the parameters of such a relationship can only be derived from extensive simulation runs. If the constant delay values are greater than the values that would be derived from execution of the models of resource availability, the ultimate effect on model outcome is poorer performance of the unit; conversely, if the constant values are less than those returned by the higher fidelity model, unit performance will improve as a result of the abstraction.

The other area in which variable fidelity is explicitly cited in the SDD deals with determining CP sites. The high fidelity model would base that decision based on ground cover and communication line-of-sight. An abstracted version of the model would have a list of pre-defined candidate sites. This abstraction is a modification of model form, in which a table look-up is used in place of an actual computation. In this case, the impact of the abstraction is that a less than optimal site may be selected from the available sites in the table compared to a direct calculation. Again, the derived effect of this abstraction is not easily quantified in terms of unit performance (vulnerability, ability to communicate orders to own forces, and so on).

A common theme in both these proposed implementations of variable fidelity models is simplification of the model by elimination of the factors that are used in a decision-making process. This theme is logical in the C2 submodel, in which the processing is largely of the form of generating a decision on the basis of some set of input data, then communicating that decision in the form of an order or message. For this reason, the analysis of abstraction possibilities in the C2 submodel should focus on parameter elimination abstractions for individual methods which map to specific types of decisions.

In some cases, the "factors" on which a decision is based are embedded within a KBR, not stored and passed as explicit parameters. For these cases the implementation of variable fidelity will require a structuring of the rule base to allow a more complex and comprehensive set of rules to be used for a high fidelity model, while a simpler and less comprehensive set of rules, which incorporates fewer factors, could be used in an abstracted version of the model. Although the parameters in these cases may not be explicit, such an abstraction would still be considered a parameter elimination abstraction.

Implementation of this type of variable fidelity requires a modularly structured rule base which has components which can be invoked as required for a particular simulation. The data flow diagrams in the SDD do not provide much insight into how specifically the rule bases could be organized. For example, the `evaluateSituation` function in the `monitorStatus` method takes as input rules from the `EvaluateSituationKnowledgeBase`, but there is no explicit input

which indicates the data being used as the definition of the situation being evaluated, or the plan which is being checked. It is within these inputs that parameters could be eliminated, such that a simplified set of rules in the `EvaluateSituationKnowledgeBase` could be invoked given an abstracted version of the plan and the current battlefield situation.

8.2.6 Movement Submodel

The Movement submodel is designed to support the movement of objects over the ground, through the air, and on or under the sea. The SDD does not explicitly identify multiple versions of methods reflecting multiple levels of fidelity, and generally treats the movement modeling at a fairly high level. In this section, some ideas are presented which could form the basis for abstracted versions of the baseline model to provide a variable level of fidelity within the Movement submodel. These ideas are based on state aggregation and parameter elimination.

One frequently-employed technique for simplifying movement models is to reduce the number of movement decisions and states. In general, reducing the number of intersection points for movement along a network, or reducing the number of different ground areas that a unit can traverse, can simplify movement processing. These types of abstractions fit the definition of requirement 2.1.10.1 in Table 5. In the case of the ARES Movement submodel, the `SpeedonTerrain` table contains information on the maximum speed that an object type can traverse various types of terrain. The information in the table provides speeds as a function of terrain type, time of year, and time of day. A simple state aggregation is to combine different terrain types in the table, so that an abstracted model only reflects a small number of different terrain types.

Another state aggregation abstraction could be applied to the traffic congestion algorithm. The algorithm distinguishes between states in which units are at separate locations and states in which the units are co-located, and models unit activities based on the distinction. That distinction could be eliminated, allowing units to move freely through the terrain independent of whether any other units are also traversing the same portion of the road network. The effect of this abstraction is that units would arrive at destinations more promptly than occurs in the baseline model. The advantage of implementing this abstraction is that it reduces the computational requirement imposed by the baseline model route congestion processing.

Another simple abstraction is parameter elimination, which has already been addressed in this study. For the Movement submodel, eliminating the distinction of maximum speeds for time of year and/or time of day could be applied. The value of such an elimination depends on the extent to which such factors impact the speed determination. Elimination of these parameters is based on explicit assumptions. Sensitivity analysis on the Movement submodel could also derive candidate parameters for elimination.

8.2.7 Combat Submodel

The Combat submodel computes equipment losses due to ground close combat. In addition, the Combat submodel includes allocation of forces of over multiple, simultaneous battles and the effect of close combat on the ability of objects to move on the battlefield.

The SDD identifies a major role for abstraction in the determination of the combat losses. The baseline algorithm contains multiple convergence loops which are computationally intensive. The SDD calls for a lower fidelity model and specifies that the design must accommodate alternative combat resolution algorithms (requirement 2.1.10.2 in Table 5). In this section, some approaches to abstracting the ARES Combat model are described.

The first category of abstractions to be considered includes model boundary abstractions, specifically, elimination of input parameters. The combat resolution in ARES is based on the following parameters:

- Type and number of specific equipment items;
- Duration of the battle;
- Battle width across front; and
- Postures of the two sides.

The equipment and duration parameters are fundamental in determining battle attrition, and can not be eliminated. Battle width influences the determination of whether a particular weapon system is available to engage a target. The practical effect of the battle width is to concentrate (as width decreases) the "energy" of the battle, resulting in higher attrition. The effect of eliminating battle width depends on how processing is modified. If width is eliminating by not considering availability, (which is the same as setting availability at 100%), the effect of the abstraction would be to increase the attrition of any battle. If availability was set at some nominal mean value, then the effect of the abstraction on individual battle outcomes could not be determined. The posture of the two sides is used to index a table of attrition outcomes (note the use of the table look-up abstraction in the baseline ARES model) derived from another, higher fidelity model, the Combat Sample Generator (COSAGE). Without setting the posture, attrition would be based on a smaller table which did not distinguish between various postures. Since the relationship between posture and attrition rates is based on the outcome of simulation runs and is not mathematically expressed, the effect of such an abstraction on combat resolution is not predictable.

Another category of abstraction techniques includes model behavior abstraction techniques. For example, state aggregation can be used to simplify some aspects of the Combat submodel. In the preceding example, we discussed the notion of eliminating the unit posture as a parameter in calculating combat attrition. A less drastic approach is to use unit posture as an input, but reduce the number of postures that are considered. For example, the posture could be reduced to

simply Attack and Defend, still preserving some of the elements of posture in determining attrition, but potentially simplifying the model. The term "potential" is used because this particular state aggregation is of little value if the only impact on computational resources is a smaller table; however, if determining and maintaining posture and calculating attrition involves significant computation, this simplification could reduce resource requirements. State aggregation could also be applied to the type and equipment information associated with each unit. An aggregated measure of combat strength could be substituted which would not require tracking of individual weapons and equipment items.

The SDD suggests an abstraction of model form as another approach to reducing the combat processing requirements. Specifically, a Lanchester type approach is identified, although the detailed design of such an approach remains to be completed. The Lanchester approach changes the form of the model from a table lookup to a computed solution, although based on some aggregate data.

8.3 Observations and Conclusions

There are a number of general observation which can be drawn as a result of this exercise. Some of these observations relate to the ARES model, while some are relevant to the idea of using model abstraction techniques to support design and development of a model with varying levels of fidelity. The observations that relate to the ARES model specifically are presented in Section 8.3.1, the more general issues are addressed in Section 8.3.2.

8.3.1 Observations Concerning the ARES Model

In terms of drawing some general conclusions on the topic of variable level fidelity in ARES, the discussion begins with the ARES SDD. The ARES SDD identifies specific functionality for variable levels of fidelity in only two of the functional models: the Communications submodel and the ISR submodel. The Combat submodel also identifies a requirement for modularity that could support multiple levels of fidelity in the combat attrition calculation. In the design of the other functional models, the issue of multiple levels of fidelity is not explicitly addressed. While the specified functionality for the various levels in these submodels is reasonable, these are certainly not the only (or even the most logical) places in which variable levels of fidelity can be achieved in the ARES model.

A more comprehensive approach to variable levels of fidelity should start with the most fundamental entity in the model, the ground unit. A similar argument applies to air units, although there are not as many levels of fidelity and opportunities for dynamic level switching. The most appropriate abstraction technique, or set of techniques, for providing variable levels of fidelity involving ground units is entity aggregation, as described in 8.3.2. There are a number of possible approaches to actually implementing such an abstraction, one of which is suggested in Section 8.2.2. Other entity aggregation schemes are possible as well. The key requirement in maintaining the validity of the entity aggregation is a consistent representation/result across

multiple echelon levels. As the relative reference point approach described in Section 8.2.2 suggests, an abstraction using a generalized approach in which the specific differences between echelon levels are isolated to the C2 procedures can be more easily demonstrated as a sound abstraction.

Entity aggregation within the Ground submodel can result in abstractions in other submodels as well. For example, the combat resolution among division level units may be different than among smaller units. One reason is that combat resolution algorithms that rely on the average effects of large numbers of individual combat outcomes do not scale down well to critical combat involving a small number of entities (see the SDD, Section 14.2.1). Abstractions in the Movement submodel will also be appropriate if division-sized ground units are used, rather than smaller units. Terrain information can be consolidated, since terrain features relevant to smaller units may not be relevant to larger units.

The remainder of Section 8.2 summarizes a number of other potential abstraction opportunities. They generally fall into two categories: parameter elimination and state aggregation. There are a number of issues that must be considered to determine whether any of the suggested abstractions would be feasible and useful for the ARES model, including the following:

- What is the actual effect of eliminating the parameter or implementing aggregation on model outcomes?
- What are the savings in terms of resource requirements?

Both questions should be answered as part of the detailed design process. The detailed design phase is the appropriate point at which code specifications of alternative implementations of model functionality can be generated and compared to verify the abstractions.

8.3.2 Observations on the Use of Abstraction Techniques

In addition to the direct application of techniques to the ARES design, this exercise of analyzing ARES also yielded some useful lessons about the abstraction techniques developed under this effort. In this section, these lessons are discussed.

First, note the extent to which abstractions can be derived strictly from design documentation. In previous analysis exercises, existing code was analyzed in some cases to identify specific abstractions. For example, it is difficult to recommend explicit assumptions that form the basis for parameter elimination without starting from the list of input parameters. That information was not available in the ARES SDD. The ARES SDD presents the design at an appropriate level for its stage in the system development cycle. However, identification of abstraction techniques is a cyclic process which should be iteratively refined and which expands in level of detail in concert with the expanding level of detail in a system design through the course of a system development. We have made some observations in this report which can impact the design at its current level. But we have also provided some ideas and suggestions of

areas where abstractions could be applied as more detail is added to the ARES system specification as part of the design process.

The ARES model also provided a useful illustration of the application of the ideas of abstraction techniques to the specified functionality of variable fidelity models. By mapping the design definition to previously identified abstraction techniques, an evaluation can be made as to whether the lower fidelity model is a reasonable simplification of more detailed model. The evaluation of ARES in this regard is generally favorable, although the analysis of the Communications submodel and ISR submodel include some suggestions to further enhance the relationship of the various levels of fidelity.

Given the importance of multiple levels of fidelity in the overall characteristics of the model, the instances in which multiple levels of fidelity are actually identified in the SDD are modest, and without apparent rationalization. In particular, no broader comprehensive perspective of multiple levels of fidelity is presented in the SDD. The specified multi-level functionality is isolated in only two of the submodels, with a reference to a possible implementation in a third submodel (Combat). It is not clear from the design documentation that these lower level fidelity models will actually result in significant computation requirements reductions, because they are limited in scope. A broader set of abstractions with broader scope which cut across submodel boundaries is much more likely to accomplish significant computational requirements reductions. The key is providing multiple levels of fidelity in the most fundamental elements of the model. In the case of the ARES model, that fundamental element is the unit, and the most effective abstraction in such a case is abstraction by entity aggregation. The advantage of such an approach is that additional simplifications can made in almost all other areas of the model, providing a comprehensive run-time reduction. The major disadvantages are the difficulty in defining valid relationships among model levels.

The various approaches discussed in Section 8.2.2 illustrate another lesson learned in the course of this analysis. Abstraction techniques are not unique. In this particular case, there are several potential approaches to aggregating entities along the military hierarchy structures, all of which fall into the same general abstraction category. Similarly, elimination of a parameter could result in several different ways of modifying the model code to delete the impact of the parameter.

9. Application of Abstraction Techniques to C^{3}I Simulation

The preceding sections provide examples of how abstractions can be applied to these specific C^{3}I models. However, because of the specific nature of the models, there are some abstraction techniques described in Section 5 which are not illustrated in the context of RWWM, ALARM, or ARES. In this section, we provide some additional examples of how abstractions can be applied to C^{3}I models, with particular emphasis on techniques that have not already been illustrated.[11] Examples of model boundary modification techniques (Section 9.1), behavior abstraction techniques (Section 9.2), and model form abstraction techniques (Section 9.3) are included in this section.

9.1 Model Boundary Abstraction

The first abstractions to be discussed are those that involve changes to the model boundary. The organization of this section follows that of Section 5.1. For each category of model abstraction techniques, we discuss examples of the application of the technique to the C^{3}I modeling domain.

9.1.1 Explicit Assumptions

The first category of abstractions to be considered involves explicit assumptions. Model hierarchies and delimited input ranges are discussed below.

9.1.1.1 Model Hierarchies

The approach of using model hierarchies with explicit assumptions within the C^{3}I domain is most useful in models that deal with the more physical or phenomenological aspects of C^{3}I (such as detection and communications). We have already discussed how the ALARM model can be structured as a model hierarchy with explicit assumptions (Section 7.2.1). Even in the ARES model, the examples of hierarchies based on explicit assumptions (such as those discussed in Section 8.2.2.1, or some of the abstractions in the Communications submodel) involve the physical processes represented in a theater level context.

A hierarchical approach provides an alternative to the often-used approach of building a single model to accommodate "all possible" simulation requirements. Such a model meets the (computationally) worst case requirements, but then requires resources to perform unnecessary computation when the simulation requirements are not as demanding. The philosophy of the hierarchical approach is to recognize a priori that there is a range of potential simulation

[11] We use the term C^{3}I models here to reflect the type of models typically used by Rome Laboratory. The key characteristics of C^{3}I models that must be accommodated in this analysis are: discrete event models with non-numeric computation (i.e., cannot be directly expressed mathematically). One could substitute discrete event models for C^{3}I models throughout this section with equal applicability. Examples are selected from the C^{3}I domain to suggest the relevence of the discussion to specific issues associated with C^{3}I modeling.

```
detect_target (target_rotor_flag, jamming_flag, clutter_flag, noise_flag)
target_rotor_flag, jamming_flag, clutter_flag, noise_flag, target_detected: Boolean
begin
        calculate basic target signal strength;
        if target_rotor_flag
                then calculate target rotor signal strength;
        if signal-interference > threshold and jamming flag
                then calculate jamming effect;
        if signal-interference > threshold and clutter_flag
                then calculate clutter effect;
        if signal-interference > threshold and noise_flag
                then calculate noise effect
•
•
•
end.
```

Figure 16, Implementation Technique for Model Hierarchy with Explicit Assumptions

requirements, and certain aspects of the model can be abstracted depending on the data to be generated.

As shown in the ALARM example, a model hierarchy need not involve separate simulation modules. In the case of ALARM, there are some examples of nullifying parameters which allow run time execution of abstractions. It is also possible to implement model code to facilitate hierarchies with explicit assumptions. For example, the ALARM model could be structured as a series of target detection computations which are selectively invoked based on model simulation requirements, as shown in Figure 16. The individual model components are executed only if appropriate flags are set. The comparisons to a threshold allow computation to be ignored if the output will not change. A computation which reduces the signal-to-interference ratio, for example, when the ratio is already below a detectable threshold will not change the procedure output.

We have thus far emphasized model hierarchies in our discussion of explicit assumptions, in part because models often have a natural hierarchical structure. The System Entity Structure (SES) of Zeigler (1984) is an example of a formalization of a hierarchical structure, in which the explicit assumptions are generally decompositions and specializations. However, a hierarchy may not be the right structure if decompositions are associative. For example, Zeigler uses an example of the economy, which can be decomposed along sectors or along geographic levels. However, the potential associative relation of the decompositions is not addressed. For example, is the agricultural sector of the regional aspect of the national economy the same as the regional aspect of the agricultural sector of the national economy? If so, a strict hierarchy is not as

appropriate as a more generalized graph structure, such as the Graph of Models (Addanki, Cremonini, and Penberthy 1991).

The full capability of the Graph of Models concept is not completely transferable to typical discrete event simulation models. It was specifically developed to support a search through a model base of qualitative models for the simplest model that meets simulation requirements. The assumptions can only be stated where the parameters have a monotonic relationship. Although some parametric relationships in the C^3I domain are monotonic (e.g., probability of detection as a function of emitted signal strength), other relationships are more complex, and therefore an automated model space search is infeasible.

However, the Graph of Models approach does have some potentially useful attributes that can be used in abstracting C^3I models. First, the idea of organizing related models in a graph structure would facilitate model libraries. The graph structure itself is a more general representation of model relationships than a strict hierarchy, and capturing the nature of the abstraction relationships could assist a model developer and/or user in performing a manual search of the model space. Second, the explicit definition of model relationships and the assumptions on which models are based requires a level of discipline not always applied in the C^3I modeling and simulation domain. Third, the information on model relationships could be used for implementation of partial model space search algorithms.

Model hierarchies are also implemented in situations in which two or more models of varying scope and fidelity are linked together. This approach is often undertaken to link existing models (such as linking a detailed sensor model into a broader command and control model, or linking an air combat model to handle combat resolution within a broader air campaign model). The representation of the detailed process is often modeled more abstractly in the broader scope, less detailed model. However, the explicit assumptions of the abstracted version of the process are often missing. A definition of the explicit assumptions, stated in terms of impacts on output variables, can help establish the traceability among integrated model components.

9.1.1.2 Delimit Input Ranges

The practice of specifically bounding input ranges to limit the applicability of models is well established. We add the notion that range limitations can be used as an explicit abstraction technique. Models that are constructed to modularize behaviors over distinct ranges of input values can be abstracted more easily when constraints on the input values are able to be determined.

9.1.2 Derived Abstractions

The second set of model boundary abstractions is characterized by approaches in which the determination of exogenous variables is derived from the original model. Approximation and selection by influences are included in this category.

9.1.2.1 Approximation

Approximation techniques are a traditional approach to reducing the computational requirements of a model. In this section, we discuss two techniques that have been developed specifically for qualitative simulation, and consider issues in applying them to C^3I models.

9.1.2.1.1 Causal Approximation

Causal approximation relies on analysis of the causal relationships of variables in the model. The PDGs shown for ALARM (Section 7.2.2) show the results of manual derivation of causal dependencies. Because the manual process is time consuming and prone to error, the real power in causal dependency techniques comes from automatic derivation of the dependencies. Although Iwasaki (1988) has reported techniques for generating such information from qualitative models, they are prohibitively expensive (Nayak 1992), and therefore unlikely to be scaleable to quantitative C^3I models. Current research on more feasible techniques for performing the causal analysis may push this area forward.

9.1.2.1.2 Model Sensitivity Analysis

Sensitivity analysis is an integral part of modeling and simulation methodology. Understanding the relative importance of the exogenous inputs to the simulation outputs is important for data collection and model validation, verification, and accreditation. Existing sensitivity analysis techniques can also be used to determine parameters which can be eliminated to abstract a model.

Weld's work (described in Section 4.4.3.2) goes beyond simply determining the relative importance of individual inputs by establishing the conditions under which the validity of the results are preserved. The concept of a fitting parameter provides a means to bound the deviation of the abstract model outputs from the original model outputs to assure the validity of the results.

As with the other work in qualitative simulation and reasoning about physical systems, Weld's approach is specifically focused on models that can be expressed in terms of algebraic and ordinary differential equations. Because the C^3I models of interest to Rome Laboratory do not generally fit this category, the algorithms described by Weld (1992) cannot be directly implemented for such models. However, we can still adapt these general concepts. Section 7.2.3 contains a discussion of how such techniques could be applied to ALARM. Since the discussion of ALARM does not address this issue in detail, we consider the application of model sensitivity analysis to a hypothetical C^3I example.

We start by considering the general concept of a fitting approximation. Consider the example of a model used to determine line of sight between two entities in the battlefield. Suppose that we start with a model that detects line of sight as a function of the following:

- The intervening terrain between the two entities;

- Intervening foliage;
- Obscurants (smoke, fog, etc.); and
- The viewing height/altitude of each entity.

One of the questions to be addressed in model abstraction is whether a functionally equivalent (i.e., one that also computes line of sight) but lower fidelity model can be abstracted by creating a fitting approximation of the model. The fitting approximation would identify one of the four input variables as the fitting parameter. Consider the analysis to determine if the viewing height/altitude is a fitting parameter, and a model based on intervening terrain, foliage, and obscurants is a fitting approximation. Using Weld's notation, we define the following:

P	is the more abstract, lower fidelity model.
BEHAV (P, v_1, ..., v_3)	is the results of that model.
Q	is the original model.
BEHAV (Q, v_1, ..., v_3, v_f)	is the results of the original model.
v_f	is the fitting parameter (height/altitude).

Note that in our example the result of the model is a Boolean value (i.e., whether there is or is not line of sight from one entity to the other). We extend Weld's concept of parametric output to include Boolean results by defining BEHAV (P, v_1, ..., v_3) and BEHAV (Q, v_1, ..., v_3, v_f) as the probability distributions associated with the resulting Boolean value as a function of the input parameters v_i.

The height/altitude parameter is a fitting parameter if the differences in the predicted behaviors approaches zero as the initial value of the fitting parameter goes to an approximation limit, i.e., if

$$\lim_{v_f \to l} \text{BDIFF (BEHAV } (P, v_1, ..., v_3), \text{ BEHAV } (Q, v_1, ..., v_3, v_f), 0, t) = 0$$

where l is the approximation limit of the fitting parameter v_f. In this example, the height/altitude parameter approaches a limit of 0 (measured as distance above the terrain level of the entity). In this example, this relationship holds, since the difference between the model prediction of Q and P would clearly approach zero as the height/altitude approaches 0.

Another required extension to Weld's definitions of model sensitivity analysis involves interpretation of the notion of the sign of the partial derivative of the fitting approximation. According to Weld, to compute the difference in the behavior predicted by eliminating a fitting approximation, one can simply follow the partial derivative with respect to the fitting parameter away from the approximation limit. Following this approach, we conclude that the effect of eliminating the fitting parameter (which is abstracting the height/altitude assuming that it is

independent of the other input parameters) is that P consistently underestimates the existence of a line of sight relative to Q.

Query-directed simplification can then be used to determine whether a particular approximation is adequate for a particular application. According to Weld, the trick is to use an inequality whose truth is being queried to select an approximate model, or to determine whether an approximate model is adequate. Query-directed simplification involves computing the bounds of the difference between the approximation and the results of the more detailed model. In this example, if entities can be airborne, then clearly the difference between BEHAV (P, v_1, ..., v_3) and BEHAV (Q, v_1, ..., v_3, v_f) will exceed reasonable tolerances. On the other hand, if entities are limited to ground based entities, so that height is limited to the height of the observer and/or the equipment, then the differences between model behaviors could be well within an acceptable tolerance. In this case, the more abstract model could be used.

9.1.2.2 Boundary Selection by Influences

Boundary selection by influences is based on model space search algorithm. Models are distinguished based on the categorization of variables in the model as exogenous, dependent, or irrelevant. The search algorithm relies on the causality relationships among variables to determine the simplest model that meets simulation requirements. There are several issues in applying such techniques to C^3I models. One issue is analysis of the causal relationships among variables, which we already noted in Section 9.2.1.1. Another difficulty is the potentially large model space to be considered. In simplistic terms, the number of models is n^3, where n is the number of variables in the model. These challenges suggest that boundary selection by influences is not an immediate candidate for application to the C^3I domain.

9.2 Behavior Abstractions in C^3I Models

The purpose of this section is to discuss how the behavior abstraction techniques described in Section 5.2 of this report can be applied to C^3I models. The behavior abstractions in qualitative simulations are based on generating all possible states. Typically C^3I models have an infinite state space (or at least an extraordinarily large state space) that cannot be totally generated. However, from the model design perspective, we can still consider analogous abstractions that achieve similar goals as the behavior abstractions described in Section 5.2 (even if they must be built into the model). The organization of this section follows that of Section 5.2; each of the techniques described there is analyzed with respect to the C^3I modeling domain.

9.2.1 State Aggregation

We identified four different approaches to state aggregation in the taxonomy presented in Section 5: behavior aggregation, causal decomposition, aggregation of cycles, and numeric representation. Each of these approaches is discussed in a subsection below.

9.2.1.1 Behavior Aggregation

The general notion of behavior aggregation appears to fit a variety of C^3I applications. We have already shown examples from RWWM (Section 6.2.2.1) and ARES (target identification and reporting in the ISR submodel, described in Section 8.2.4) of behavior aggregation.

However, in the discrete event domain, without an envisionment of the behavior states, there is not yet an automated approach to determining how behaviors can be aggregated, so the behavior aggregation opportunities must be identified manually. The principle of single input single output (SISO) transition graphs can be applied without necessarily generating an envisionment graph.

One approach is to identify the system states that reflect the distinction in simulation requirement-driven output values, and determine the state trajectories that generate those distinctions. State trajectories that do not result in any distinctions are then candidates for aggregation.

9.2.1.2 Causal Decomposition

The notion of causal decomposition could be applied to C^3I models. We discuss a specific example in Section 7.2.5. The causal relationships of parameters in the model are determined a priori, and that information can be used to abstract system states. We can describe this situation notionally in the following manner. Suppose there is a model in which parameter B is causally related to parameter A; that is, B is functionally dependent on A. Further, suppose that the question being addressed by the simulation is the outcome of parameter B. System states which only differ in the value of A but not B can be aggregated to a single state.

For example, suppose there is a model in which the simulation requirement is to determine probability of kill (P_k), and the P_k is a function of range, with a P_k of 0 when range exceeds some threshold. P_k is causally related to range. The system states with the range which exceeds the threshold can all be aggregated to a single state, because the system state differs only in the value of the range (the value of P_k is 0). The a priori knowledge of causality could be exploited by initially checking whether the weapon can possibly be moved into range; if not, more detailed computations are necessary.

9.2.1.3 Aggregation of Repeating Cycles

The initial concept of aggregating repeating cycles, introduced by Weld (1986), is based on manipulations of a total envisionment. In the case of discrete event simulations that feature a single sequence of states, identification of repeating or similar cycles would require maintaining a history of states and then checking to determine if the state had previously been achieved. This could be feasible in some specific cases in which an iterative situation is modeled and can be abstracted; however, the maintenance of state history information could also become very cumbersome in many models.

However, a priori knowledge of the iterations in the model could be used to implement abstractions of repeating cycles, as we showed in the example using RWWM (Section 6.2.2.1.1.2). We can generalize the RWWM example to other situations in which two entities are moving, and their separation distance is repeatedly checked to determine whether they are within range. The repeating cycle is an update of their location and a check of the separation range. This sequence of states could be aggregated with a single state (separation distance exceeds threshold) as follows:

1. Calculate mathematically (given each entity's current location, velocity, and acceleration) the simulation time when the entities will be in range. Note that this calculation could identify situations in which the entities will not close to within the threshold.
2. There is a state change (i.e., an event is processed) if there is a change to the entity's acceleration. The result is a new system state with new entity parameters, but still with separation distance exceeding threshold.
3. If the entity parameters do not change, there is a state change (i.e. an event is processed) at the calculated time.

9.2.1.4 Numeric Representation

Numeric representation is another example of a technique that can be considered to already be part of the model developer's arsenal, although the application is often motivated by resource considerations rather than simulation requirements.

9.2.2 Temporal Abstraction

The mechanics of applying temporal abstraction to C^3I models are straightforward; for unit advance simulations, the time advance unit is increased. The issue, as always, is ensuring that the validity of the results is not compromised. Analysis of the state dependency to time scale can address the validity question. Alternatively, a model sensitivity experiment can be conducted to determine model sensitivity to time step.

Temporal abstraction within a model also arises as a specific challenge in the C^3I modeling environment because of the disparity in activity tempos among various entities. For example, this issue arose specifically in the integration of ground, naval, and air combat activity models in development of the Joint C3 Simulator, described by Frantz, Gillmore, Brown, Foreman, and Pelkey (1990).[12] The component models have processes that are relevant at different time scales. For example, communications is nearly instantaneous, while fighter aircraft are moving across the battle space at a rate of hundreds of kilometers per hour, but ground and naval vehicles are moving at speeds one to two orders of magnitude slower than the aircraft. To answer a question about the capabilities of the communications systems, a small (fast) time scale is appropriate,

[12]Specifically, see Section 5.2.2. of that report.

such that variables such as vehicle locations can be held constant. At the other extreme, to answer a question about the movement and behavior of a ground vehicle, it may be appropriate to abstract aircraft movement, and simply have the aircraft appear in the scenario at the appropriate time rather than model its entrance in detail.

9.2.3 Entity and Function Abstraction

Entity and function abstraction have analogs in discrete event simulation. Entity abstraction involves replacing models of lower level entities with a model of an aggregated entity. Abstraction by entity aggregation is a commonly used abstraction, particularly in models of military activity. Within the ARES Communications submodel (see Section 8.3.3), networks are aggregated to abstract the representation of the communications entities. Likewise, in the ISR submodel (see Section 8.3.4) entity aggregation is applied to the sensors to create lower fidelity versions of the model. Entity aggregation refers to the representation of a "higher level" entity to represent a collection of "lower level" entities.

The use in military simulation is due to the inherent hierarchical structure of militaries that often drives simulation modeling requirements. Hierarchical structures are recommended as the basis for ground unit entity abstractions in ARES.

Entity aggregation is often accompanied by other aggregation as well. The appropriate time cycle for a model of division level activity is not the same as the time cycle for an individual tank, so temporal aggregation would be appropriate. Likewise, the domain of potential system states for a division level model is different than the domain for the individual collection of tank models, leading to behavior aggregation.

9.3 Model Form Abstractions in C^3I Models

The third category of model abstraction techniques includes those which modify the form of the model. As described in Section 5.4, these abstractions include look-up tables, probability distribution functions, linear function interpolation, and metamodels. The first three techniques are frequently used standard techniques for C^3I model abstraction and are not further discussed. Although not as mature, metamodeling as an abstraction technique has been demonstrated specifically in the C^3I domain on a radar model entitled TERSM by Zeimer, Tew, Sargent, and Sisti (1993) and Caughlin (1994).

10. Advanced Concepts

The preceding four sections provide numerous examples of how abstraction techniques can be applied to actual C^3I models. In addition to providing illustrations of the techniques, the exercise of applying these techniques to actual models revealed some areas requiring additional thought. There were several situations in which the taxonomy introduced in Section 5 was insufficient to fully deal with model abstractions. In particular:

- No formal definition of abstraction techniques exists that can be used as the foundation of associating mathematical attributes to the techniques;
- In some situations, an abstraction technique could not be efficiently implemented without making some additional simpifying assumptions in the model; and
- Identifying candidate abstraction techniques can be difficult given a large, complex model.

Concepts to address each of these issues are discussed in a subsection that follows.

10.1 Formal Definition of Abstraction Techniques

Recall that one of the motivations for studying abstraction techniques is to facilitate validation by formalizing the process of moving from one conceptual model to another. Formalizing this process requires a rigorous, formal definition of the abstraction techniques. There are at least three formalisms for discrete event simulation that could provide the basis for abstraction definitions: the Discrete Event System Specification (DEVS) described in Section 3.3 and Appendix A (Zeigler 1984); the formalism described by Ho (1989); and the formalism developed by Radiya and Sargent (1994). We did not determine the formalism that provides the best basis for defining abstractions, although the DEVS formalism has the advantage of being more mature and having a larger experience base within the community. To illustrate the issues associated with formal definition, we use the DEVS formalism in the remainder of this section.

Within DEVS, the mechanism for expressing relationships between models is the morphism. Morphisms are transformations that preserve specific properties of a model specification. The concept of morphism deals with specifications of models, rather than models themselves, whereas abstractions are applied to the models themselves. Although subtle, this difference means that abstractions cannot be defined specifically in terms of the DEVS morphisms. The morphisms defined by Zeigler (1984) that cross levels of resolution introduce concepts in which the levels of resolution are defined in terms of a structure-behavior axis. These morphisms allow manipulation of model structure to be abstracted to its behavioral essence. Other approaches to model abstraction (such as those addressed in Section 5.3) tend to deal with structural or behavioral abstractions, rather than moving from one level of specification to another.

The levels of system specification in DEVS are narrowly defined, so that abstractions (associations) across specification levels have a limited domain. However, the concept of morphism may be useful as a measure of the preservation of model features. One issue to explore is whether a particular abstraction preserves features at a given level. For example, a particular abstraction may result in a model that is isomorphic to the original model at the I/O Relation level. Such a metric would be useful in assessing the utility, applicability, and validity of a specific abstraction with respect to specific simulation requirements.

Another issue requiring further exploration involves the utility of DEVS in expressing real-time aspects of C^3I models. Discrete event systems in the DEVS context are defined as systems driven by events. Events are instantaneous changes in variable values that may occur at irregular time intervals, but whose timing is known in advance. Also, the output of a component is defined at the point in time "just before" it makes its next state change. However, event occurrences are unpredictable in real-time simulations, and timings are not necessarily known in advance. The implications of these attributes of DEVS for defining abstractions to real-time simulations remain to be determined.

10.2 Primary and Secondary Abstractions

In the analysis of RWWM, ALARM, and ARES, there are examples of model transformations that could be categorized as different abstraction techniques. For example, in the RWWM model, the abstraction of the maneuver computation involved the elimination of the turning radius parameter. This transformation could be considered either a model boundary abstraction or a state aggregation. In many cases, a given model transformation does not belong exclusively to a single element of the taxonomy presented in Section 5. A transformation can involve an entity aggregation and a temporal aggregation, or an approximation and a temporal aggregation. Also, the mappings of the techniques reported in the literature to techniques defined in the taxonomy are not always exact. For example, the Compositional Modeling described by Falkenhainer and Forbus (1991) generally fits our definition of a model boundary abstraction with explicit assumptions. However, the grain assumptions within Compositional Modeling fit our concept of entity aggregation.

The abstraction techniques defined in Section 5 are not mutually exclusive. It is not necessary to distinguish a particular model abstraction as being uniquely entity or state or temporal or process aggregation (or some other category that provides the mutual exclusivity). Rather, we focus on determining the relationships among aggregations that can be exploited to benefit the model developer.

For example, under what circumstances does entity aggregation require temporal aggregation? Under what circumstances is it not mandatory, but highly desirable? Applying the logic of Iwasaki (1992) provides some ideas of qualitative analysis of this issue. If the processes of the detailed entities generated behavior in a time cycle that appears discontinuous when the aggregate entities are simulated, then temporal abstraction is highly desirable.

As another example, consider the relationship between state aggregation and entity aggregation. Entity aggregation is an abstraction that is often considered in C^3I models (see Section 8.3.2). By definition, the domain of system states changes as entities are aggregated. The data maintained for a model of individual combat vehicles is clearly not the same data as that maintained for a model of a combat platoon. What is less clear is the mapping from the state space in the initial model to the state space of the aggregated entities. The mapping could involve state aggregation, in which several states are mapped to a single state. In other cases, new states may be defined that approximate a set of system states. The nature of this mapping is important not only for ensuring an efficient implementation of the abstracted model, but also for model validation.

Transformation of a model may also involve abstractions that include both boundary and behavior modifications. The example cited above of the maneuver computation in RWWM fits this category. Parameter elimination can also lead to other abstractions such as temporal abstractions. For example, a parameter that is eliminated because the model is relatively insensitive to it may be the only aspect of the model that causes (in an event advance simulation) or requires (in a unit advance simulation) state changes in small time increments. In this case elimination of the parameter requires temporal aggregation as well.

In the examples described in this section, in cases where a model transformation involves multiple abstraction techniques, one technique stands out as the primary approach. Then, other aspects of the model are abstracted as a result. We describe the initial change to the models as a **primary abstraction**, and the related changes to the model as **secondary abstractions**.

Being able to identify an abstraction as primary or secondary is not the major issue, since model transformation is a model transformation regardless of how it is defined in terms of the taxonomy in Section 5. The real value of this concept of primary and secondary abstractions is that it describes a phenomenon that we have observed in model development efforts. Primary abstraction techniques are often applied without considering the corresponding secondary techniques. This approach can lead to situations in which the time scale is no longer appropriate for the entities being modeled, or there is detail generated by one component of the model which is no longer relevant for the model, or there are input parameters which are no longer needed by the model. At least, these situations involve model development, execution, and validation inefficiencies; at worst, they could result in incorrect models.

We have made some simple observations about secondary abstractions from the analysis of RWWM, ALARM, and ARES:

1. If an input parameter is eliminated and it does not have a nullifying value, then a secondary abstraction of the behavior which that parameter influences is required.
2. If a parameter which causes the most time sensitive changes in system state is eliminated, then a temporal abstraction is appropriate.

3. If the processes of a detailed entity's generated behavior appears discontinuous when the aggregate entities are simulated, then a temporal abstraction is appropriate.

These rules are merely a start to an appropriate set of rules for identifying and implementing secondary abstractions, and they are based only on our observations. An appropriate topic for future research is to define and formalize a more comprehensive set of such rules.

10.3 Surrogate Abstractions

The exercise of analyzing existing models for abstraction opportunities provided some valuable lessons in realistic application of the abstraction techniques described in this study. While we were able to identify some potential abstractions, it was also clear that for large and complex discrete simulation models, it is difficult to identify potential abstractions directly from model code or model documentation. Thus an important element in the utility of abstractions is a means of efficiently determining how to apply them.

Given the complexity of models, the key is to find an alternative representation of the model for which abstractions can more easily be identified. The multi-disciplinary focus of this study provides the ideas for different representations of a model as a means for identifying candidate model abstractions. Specifically, we propose the idea of **surrogate abstraction techniques**. By transforming the model into a different (surrogate) domain, we may be better able to determine how the model can be simplified.

The purpose of this section is to further explore this concept of surrogate abstraction techniques and domains. We begin this section with a discussion of the concept of surrogate abstraction techniques. In Section 10.3.2, we discuss metamodels as a surrogate domain, and relate this work to previous and ongoing work in the application of metamodeling to C^3I models. Section 10.3.3 includes a discussion of qualitative models as a surrogate domain.

10.3.1 The Concept of Surrogate Abstraction Techniques

The concept of surrogate abstraction techniques is portrayed in Figure 17. We begin with a discrete event simulation model M. Suppose that a transformation θ can be defined from a discrete event simulation model to a metamodel or a qualitative model, resulting in θ (M). Then abstraction techniques (shown as α), such as model sensitivity analysis and behavior aggregation, could be applied to the "surrogate" version of the model, resulting in an abstract version of the model in which certain behavior properties are preserved, or deviations in behaviors are bounded within a certain tolerance. The same abstractions (α) could then be applied to the discrete event model, resulting in a discrete event simulation model in which the same behavior properties are preserved. Clearly the key to successful application of surrogate techniques is identification of a domain which facilitates both the mapping of the model into the domain (θ) and the application of abstractions within that domain (α). We have identified two domains in which there exists a baseline in developing the mappings and abstractions: metamodels and qualitative models.

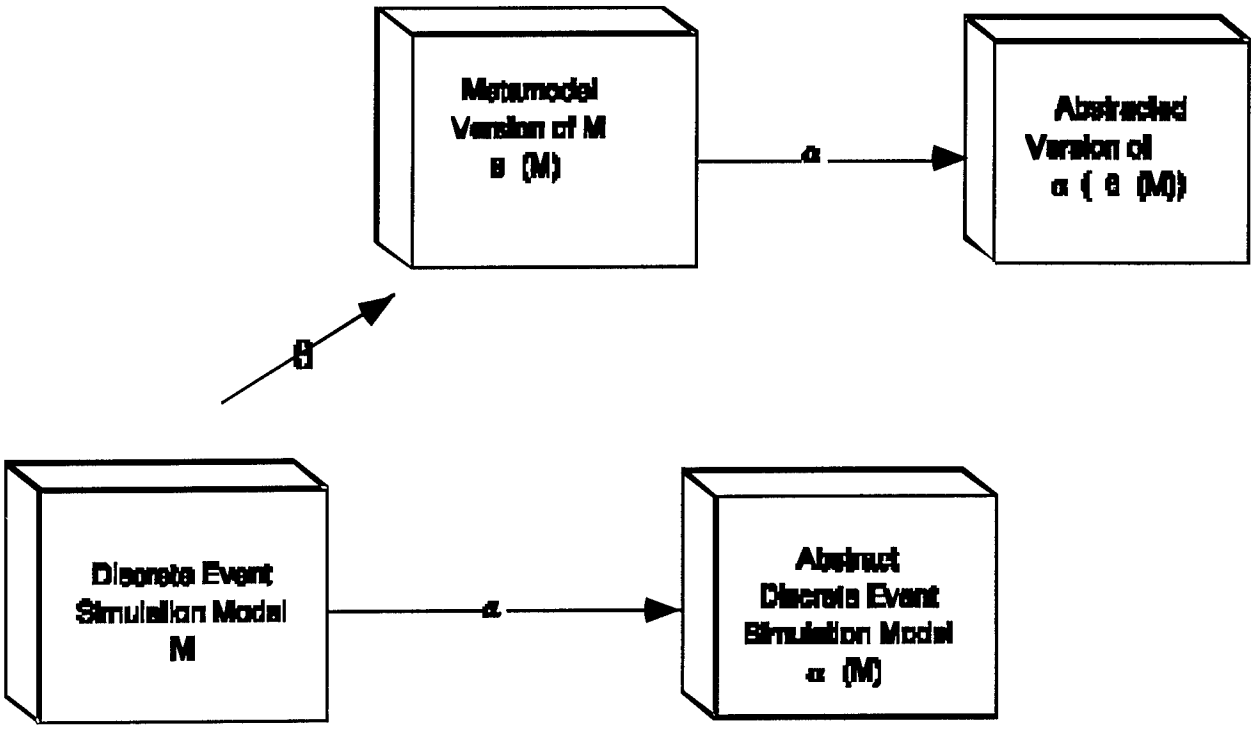


Figure 17, Surrogate Abstraction Concept

Metamodels are mathematical expressions which approximate the input/output transformations of a model. Although still an evolving area, there is a substantial and growing body of research on the mapping of discrete event simulation models to metamodels (our θ transformation). The key to using metamodels as a surrogate abstraction domain is to determine the abstractions that can be derived from metamodels. Since metamodels are already used for (among other applications) performing sensitivity analysis, abstraction by parameter elimination is a logical abstraction for which metamodels could be used. We discuss metamodels as a surrogate abstraction domain in Section 10.3.2.

Qualitative simulation also provides a potentially rich set of abstraction techniques (our α transformations) to be applied in the discrete event simulation domain. The key is the definition of the qualitative transform θ, but little work has been done in this area thus far. Issues concerning qualitative models as a surrogate domain are addressed in Section 10.3.3.

10.3.2 Metamodels as a Surrogate Abstraction Domain

We begin our discussion of metamodels as a surrogate abstraction domain with a brief introduction to metamodels in Section 10.3.2.1. We then turn to a specific example of the application of metamodeling to a C^3I model. Zeimer et. al. (1993) developed metamodels for a specific tactical model that had been developed and used by RL for conducting systems analyses of airborne reconnaissance systems. In the remainder of this section, we consider several aspects of these metamodels. Section 10.3.2.2 includes a discussion of the metamodel results and

provides some perspectives on the model not included in the original paper. Section 10.3.2.3 explores the relationship between metamodels and parameter elimination.

10.3.2.1 Metamodels

As defined in Caughlin (1994), a metamodel is a mathematical approximation of the system relationships defined by another, more detailed model, such as a C^3I model. It is a black-box approximation of the causal time dependent behavior of a simulation model that allows assessment of individual factors on the performance of the simulation. These approximations can be used for studying system behavior, predicting response, sensitivity analysis, and optimization.

There are two basic techniques available for developing metamodels: direct and inverse modeling. A metamodel is developed directly by applying basic principles to the original model to create a more abstract version of the model. Typically, such a metamodel preserves the structural and functional organization of the real world system that is incorporated into the original model. It allows the prediction of the response to exogenous inputs as a function of system state. Many of the techniques described in this report fit this definition of direct metamodels and have already been discussed at length, so the focus of this section is on the inverse approach to metamodeling. The term "metamodel" in the remainder of this discussion refers specifically to the inverse metamodeling approach.

The inverse approach to developing a metamodel is based on deriving a relationship between observed input/output relationships without using a characterization of the internal processes, structures, or functions of the model. Several such relationships can be derived, so that part of the development of a metamodel is the selection of the alternative relationship that best represents the original model. Since inverse derivation of a metamodel builds a model from data, it can be thought of as a descriptive model rather than an interpretive model.

One way to consider the concept of surrogate abstraction is as follows: in situations in which a direct metamodel is difficult to identify, the perspective provided by an indirect approach can be used. That insight then leads to a direct derivation.

10.3.2.2 The TERSM Metamodels

To illustrate the use of metamodels, we chose as a starting point a study sponsored by Rome Laboratory to demonstrate the application of metamodels to C^3I models. As part of that study, a set of metamodels were developed for the Tactical Electronic Reconnaissance Simulation Model (TERSM) model. The results of that study were documented by Zeimer et. al. (1993). In this section, we describe the TERSM metamodels and some observations concerning metamodels that can be illustrated using the results of that study.

TERSM was originally built in 1969 by the Rand Corporation for the purpose of making comparative performance evaluations of a variety of airborne direction-finding systems. The TERSM model simulates a reconnaissance mission through a pulsed radar environment, and

Table 7, TERSM Metamodel Input Domain

TERSM Metamodel Input Domain			
Parameter	Units of Measure	Low Value	High Value
Altitude	feet	5000	40000
Velocity	knots	186	1150
Azimuth	degrees	60	150
Channel Capacity	number of channels	4	30

outputs emitter detections and their circular errors of probability (CEPs). The metamodels were specifically developed to capture the relationship between four key input parameters (platform altitude, platform velocity, azimuth angle, and channel capacity of the receiver) and the number of emitters detected with a CEP of less than five nautical miles (the measure of effectiveness). A specific region of the input domain was selected as shown in Table 7.

These input values were centered and scaled, so that the values were each mapped to a range of [-1, 1]. Seven metamodel iterations were made, using a set of forty-nine (49) observations drawn from test runs of the TERSM model. Details on the derivation of these metamodels can be found in Zeimer et. al. (1993).

In order to better visualize the metamodels, a program was written to generate data as calculated from the metamodels. The data was then read into Microsoft Excel, which was then used to generate graphs of the data. The graphs included in this section were part of a set of graphs that were generated as part of this study. For each of the four input parameters, five graphs were generated, in which each of the remaining parameters was set to the same constant value: -1, -0.5, 0, 0.5, and 1. Data was generated for each of the seven metamodels. These twenty graphs are included in Appendix E.

Figure 18 illustrates how the metamodels can be used to provide insight into potential abstraction techniques. Figure 18 shows two metamodel graphs. The top graph depicts the portion of the response surface of the metamodels in which altitude is varied while the velocity, azimuth, and channel capacity are equal to their maximum values. The bottom graph shows metamodel outputs where the velocity is varied, while the other parameters are at their maximum value. We observe that there is clearly more variability in the model output from variations in the platform velocity than in platform altitude. From this observation, we can conclude that altitude is a more likely candidate for abstraction than velocity. Stated another way, velocity has greater impact on the final measure of effectiveness than does altitude. Thus simplification of the model by eliminating altitude as a consideration should alter the final outcomes less than elimination of velocity.

To understand why elimination of altitude is a potential abstraction, it is useful to look at the TERSM model itself. Aircraft altitude is used in the TERSM model in two portions of the calculation of emitter detection. It is used to determine the lookdown angle from the sensor to the emitter to determine if the lookdown angle exceeds a minimum threshold. Altitude is also used to determine the three dimensional distance from the emitter to the sensor as part of the signal-to-noise ratio computation.

The impact of changing altitude on these two computations has an offsetting effect on the number of emitters detected. As altitude increases, the lookdown angle increases, reducing the number of emitters that are outside the minimum angle threshold, thereby increasing the number of emitters detected. Simultaneously, as altitude increases, the three-dimensional distance from emitter to sensor increases, thereby reducing the signal-to-noise ratio and reducing the number of emitters detected.

The metamodels suggest that in the center range of altitudes, the offsetting effects are fairly comparable. As altitude is reduced to the lower quadrant of possible values, the reduction in detected emitters due to lookdown angle threshold predominates. In other words, beyond a certain point the additional number of emitters detected because of higher signal-to-noise ratio is more than offset by the fewer number of emitters within the sensor lookdown angle. At higher altitudes, the additional number of emitters within the lookdown threshold is more than offset by the emitters lost due to the signal-to-noise reduction.

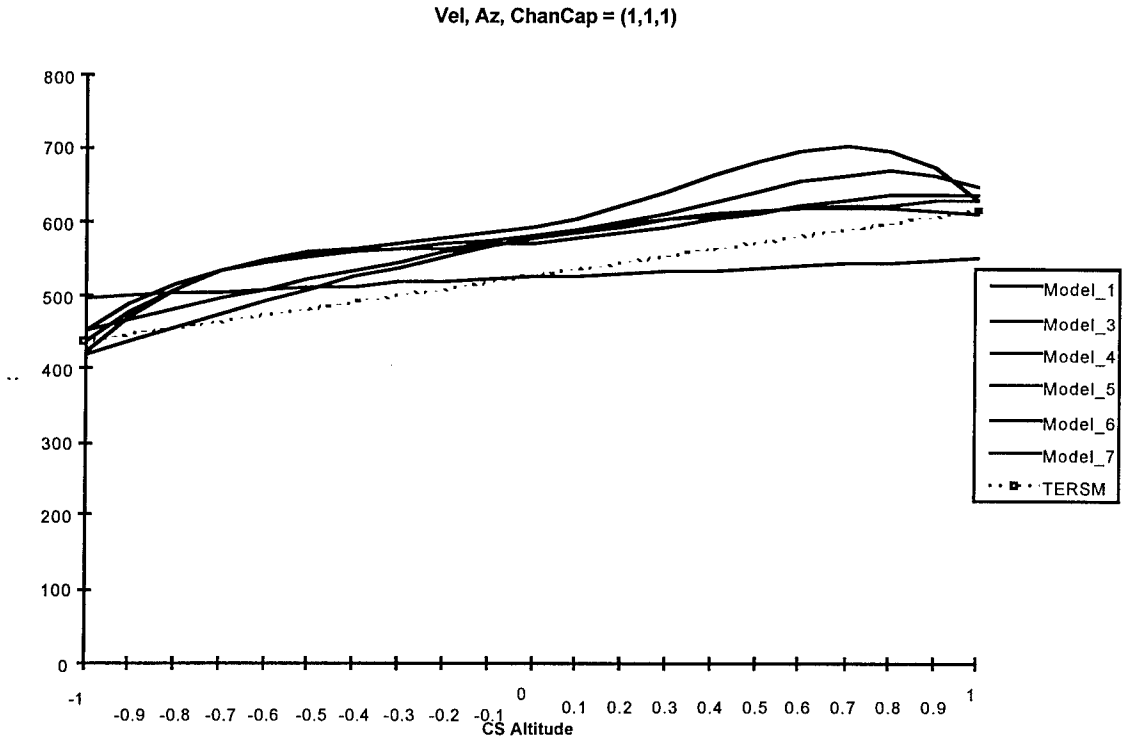
Abstraction of TERSM to eliminate altitude as an input parameter requires the following changes. First, the altitude factor would be eliminated from the three-dimensional distance calculation; instead, a two-dimensional distance calculation from the emitter to the aircraft location projected on the ground would be used. Second, the determination of emitter dropouts due to lookdown angle thresholds would be eliminated.

The value of metamodels in identifying this abstraction possibility derive from the ability to understand the impact of input parameters on the outputs without extensive generation of model data. Identifying this particular abstraction directly from the design documentation or the model code would be impossible, since the abstraction depends on the particular mission scenario (in particular, the deployment of emitters and the aircraft flight path) from which the data was taken. Also, since the abstraction is based on offsetting implications of the input parameter, it would be difficult to identify without analyzing the output data.

Metamodels are not absolutely required to identify the potential abstraction; the response surfaces of the output could be generated with a larger number of model executions. Statistical techniques such as analysis of variance (ANOVA) techniques could be used to specifically determine the contribution of individual parameters on model output. However, as models of the model, the metamodels provide a more tractable means to view the input-output relationships to facilitate the analysis of possible model abstractions. Furthermore, the metamodel provides additional information that may be useful in identifying and implementing an abstraction. For

example, for a parameter to be a fitting parameter in the sense of Weld (1992), the relationship between the parameter and the model output must be monotonic. A metamodel could provide insight in this area as well.

The metamodels also provide additional insights that may not be apparent from a statistical analysis of the data. For example, the variance in the MOE as a function of altitude appears at the highest and lowest altitudes. The metamodels reflect very little variance in detections when altitude varies over the range [-0.5, 0.5]. This information could be used to abstract the model by combining altitude states over a range in which the differences in altitude are insignificant.



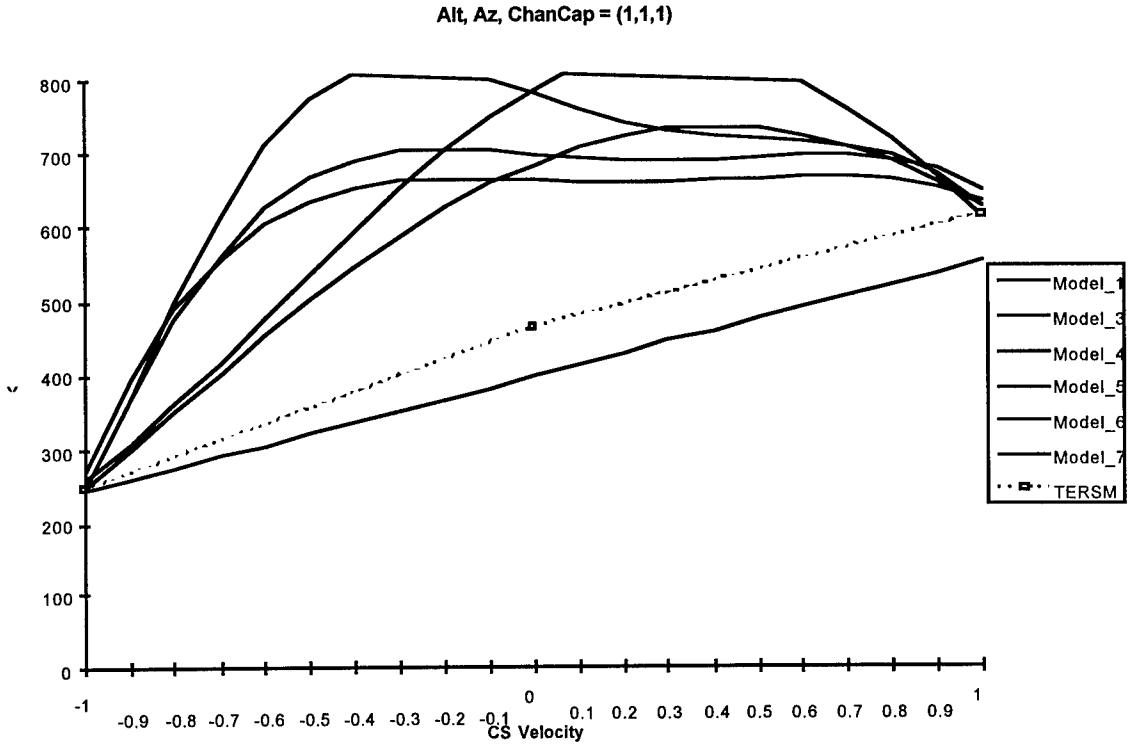


Figure 18, Comparison of Altitude and Velocity Variability Using Metamodels

This particular profile of the metamodel curves is distinct for altitude; the other parameters show different profiles. For example, the curves which reflect the relationship of MOEs to varying velocity show a significant gradient in the scaled velocity range [-1.0, -0.4], a maximum in the vicinity of -0.4, and then a more gradual descent in the range [-0.4, 1.0]. This information could be used to execute the model with a scaled velocity value of -0.4 to generate an upper bound of the output. The metamodels could also be used to identify the exact optimum velocity.

The preceding discussion is quite informal, based on graphical representations of various cuts of the data. A future research study should formalize this procedure, looking specifically at metamodels that assist in determining which abstractions to apply, and demonstrating formally the impact of particular abstractions.

10.3.2.3 The Relationship of Metamodeling and Approximation by Parameter Elimination

The concept of surrogate abstraction suggests that metamodels can be developed that provide the basis for performing sensitivity analysis. The sensitivity analysis can then identify fitting parameters for the model which can be eliminated to simplify the original model.

One must be careful in considering the composition of the metamodeling and parameter elimination functions. Suppose θ is a metamodeling function, and α is an abstraction function which eliminates an exogenous parameter in the model. Both θ and α map from the model set to

the model set, where the model set is the set of models which describe the real-world system being considered. The α function is characterized by a reduction in the number of exogenous variables in the model. Consider the differences between:

α(M), which is eliminating an exogenous variable of the model;

$\theta(\alpha(M))$, which is creating a metamodel of the abstracted version of the model; and

$\alpha(\theta(M))$, which is eliminating a parameter of the metamodel.

The first example, α(M), has already been described as the surrogate abstraction. α represents the execution of the exogenous variable elimination determined by means of sensitivity analysis on a metamodel. Now suppose that a metamodel was developed for the new model, e.g., $\theta(\alpha(M))$. Clearly this metamodel would not contain the eliminated exogenous variable, but would provide a mathematical approximation of the abstracted model and the relationships of the other exogenous inputs.

The alternative function composition creates the metamodel first, then eliminates the parameter. Eliminating a parameter from a metamodel involves eliminating any terms containing that parameter from the metamodel. Recall that the metamodels in the TERSM example are all expressed as the sum of terms which are each the product of a coefficient and one or more input parameters which have been centered and scaled to map to the range [-1, 1]. Thus eliminating a parameter has the same effect in the metamodel as setting the parameter value to 0, which is setting the parameter to a constant value in the center of its nominal range. Setting the parameter to its (approximately) mean value in some cases may be equivalent to eliminating the parameter as a factor in the model. In other cases, particularly when the parameter has a singular monotonic effect on the outcome of the model, the effect may be quite different. For example, if the effect of channel capacity is to restrict the number of targets that can be detected, then completely eliminating the parameter from the model is equivalent to using a maximum value (infinite capacity) for that parameter. Stated another way, the metamodels do not have nullifying parameters.

10.3.3 Qualitative Models as a Surrogate Domain

The second candidate approach to surrogate abstraction techniques is to exploit the various abstractions that have already been defined for qualitative models. The key issue is the mapping of a discrete event simulation model to a qualitative version of that model. To illustrate, Appendix F contains an analysis of creating a qualitative model from a discrete event model of a simple physical system.

The ability to derive a qualitative model from a quantitative model remains an outstanding research question. There has been work done in building the relationship between a numeric model and a qualitative model of the same physical system, most notably that reported by Forbus and Falkenhainer (1992) for self-explanatory simulations. Their system (SIMGEN) provides the mechanism to link qualitative and quantitative models of the same process.

SIMGEN takes as input a qualitative model and generates a total envisionment. This envisionment defines the qualitatively distinct regions of behavior of the system and the causal linkages among parameters.

The key to integrating the two types of simulations is in the definition of states, with a means for ensuring (or at least checking) the consistency of the numeric and qualitative descriptions at various states. Forbus and Falkenhainer define the concept of a hybrid qualitative-quantitative state as follows. Start with a traditional numeric state definition of a set of values for continuous processes N_f. For each non-ordering proposition class in the set of propositions that define the qualitative states, add to N_f a Boolean variable whose value expresses whether the corresponding statement is true. A state is then defined as $<N_f, Q_f>$, where Q_f ranges over the set of states in the envisionment. A state is consistent if and only if the values of N_f satisfy the propositions of the qualitative state corresponding to the value of Q_f.

The self-explanatory simulation has three major components:

1. A set of procedures which specify how the numerical parameters should be updated over time for each qualitative state;
2. A set of state transition procedures which detect qualitative changes in state; and
3. An explanation facility which uses information from the envisionment to provide causal accounts and characterize possible behaviors.

This approach still requires that a qualitative model be developed a priori. SIMGEN is then useful for establishing the linkages with the numerical model. By establishing the state correspondences between models, it is feasible to establish the correspondence between models. However, a process to derive a qualitative model remains to be developed.

11. Conclusions and Recommendations

This section concludes the main body of this report. In this section we summarize the significant accomplishments of the research and provide some perspective on the results. We also provide some ideas for future research directions which can build on these significant results.

11.1 Significant Accomplishments

The preceding ten sections provide technical details on a variety of approaches to model abstraction, and provide some specific examples of how abstractions could be applied to actual models. In this section, we step back from the detail and highlight the accomplishments of the effort, and the significance of the results in terms of the overall body of knowledge on C^3I simulation and modeling.

Each of the following accomplishments is discussed in a subsection which follows:

- Identified a number of concepts for model abstraction from outside the traditional discrete event simulation domain, specifically from the domain of model based reasoning.
- Organized a variety of model abstraction approaches into a taxonomy of techniques, which provides a structure for comparing, contrasting, and selecting techniques.
- Used models ranging from one-on-one combat to theater level combat to illustrate how the abstraction techniques could be applied to simplify C^3I models.
- Introduced several new concepts relating to model abstraction techniques.

11.1.1 Abstraction Techniques from Various Disciplines

This contract began on a premise that research in model based reasoning systems, specifically qualitative reasoning systems, had significant applicability for discrete event simulation models. That premise was initially derived from a review of Weld's (1992) model sensitivity analysis work. Further research identified a number of other approaches in the qualitative domain that were considered as well. The topicality of the research was borne out by the publication of the seminal textbook on the subject of qualitative reasoning (Kuipers 1994) during the course of this research effort. While our initial perspective on the applicability of abstraction approaches from the domain of model based reasoning was well-founded, it should also be noted that such "applicability" is thus far limited to the conceptual level. Most of the techniques are based on generation of the total, or at least the attainable, envisionment of the model, which is impossible for discrete event C^3I models. At best we developed ideas for analogous abstraction techniques for discrete event C^3I models. The idea of qualitative models as a surrogate abstraction domain, discussed in Section 10.3, is the key to more direct exploitation of

the specific algorithms developed for abstracting qualitative models. However, as described below, this is also an area which requires additional research.

11.1.2 Taxonomy of Model Abstraction Techniques

The second significant accomplishment of the effort was the creation of the taxonomy of model abstraction techniques presented in Section 5. Only one attempt has been made in the literature to create a taxonomy of abstraction techniques (Fishwick 1988), which was broader in scope and less detailed than our taxonomy. The taxonomy provides an organizing structure that is particularly important considering the fact that techniques are being drawn from multiple disciplines. By showing that Zeigler's coarsening of values attacks the abstraction problem in a manner similar to state aggregation, but different from Weld's model sensitivity analysis, we lay the foundation for comparing and contrasting various techniques.

11.1.3 Application to Actual Models

The only way to truly understand whether the model abstraction techniques can actually be applied to discrete event C^3I models is to take a model and modify it accordingly. Under this effort, we looked at three different models to ensure breadth of application, rather than concentrate on a single model in more detail (although that is a logical subsequent effort, as discussed in Section 11.2). The analysis documented in Sections 6, 7, and 8 illustrates a number of different ways in which the models could be simplified in a bounded and predictable way using the model abstraction techniques described in this report. In some cases, such as the aggregation of cycles in the RWWM launch point calculation, we were able to illustrate the impact of an abstraction in some detail. In other cases, we could only identify areas in which a particular abstraction had potential.

This analysis also led to some conclusions about the methodology of applying abstractions to existing models. For some abstractions, it is easier, or even necessary, to specifically work from the detailed design or software code level. It is at this level that specific parameters can be eliminated, or specific state aggregation abstractions can be realized. On the other hand, some abstractions are clearer in dealing with more conceptual description of the model, such as the top-level design documentation. Such examples include aggregation of entities and functions.

11.1.4 Advanced Concepts

The final accomplishment of this effort is the introduction of several "advanced" concepts that increase the utility of abstractions techniques as modeling and simulation development tools. These concepts include:

- Primary and secondary abstractions;
- Surrogate abstraction domains;
- Nullifying parameters; and

- Soundness of abstractions.

Since the concepts themselves have already been discussed in preceding sections, the following paragraphs focus on the significance of each of these concepts.

The concept of primary and secondary abstractions is important to address problems which arise when some aspects of a model are represented in great detail, while related portions of the model are not. For example, consider a model of ground force movement in which individual vehicle locations are updated once every ten seconds. If the model is abstracted by entity aggregation, the resultant abstracted model could have company positions updated once every ten seconds. The failure to recognize temporal abstraction as a secondary abstraction results in inefficiencies caused by model state recomputation in the absence of any significant change.

One of the conclusions of the analysis effort is the difficulty in identifying abstraction opportunities from arbitrarily complex models. It is to address this difficulty that we introduced the concept of surrogate abstraction domains. While requiring significant additional research to fully define, this concept provides a means to identify potential abstractions in a more structured manner. That structure is the key to moving from abstraction as an "art" based on the practitioner's experience to a repeatable process.

The concept of a nullifying parameter provides one approach to building multiple levels of fidelity directly into a model component. Careful coding allows variable fidelity to be instantiated based on the data provided to the module, rather than on changes to code.

The final concept introduced in this study is that of the soundness of an abstraction. We deliberately avoided the use of the term validity, since we use validity within the context of a specific simulation requirement. If model abstractions could be applied instantly, there would be no reason to consider abstractions that are independent of the requirement. A baseline model could be stored and retrieved in its abstracted form for each execution. However, since such a capability exceeds the current state of the art, we must instead consider model abstractions that are implemented a priori, in the form of a set of models in which the appropriate abstraction is selected for the particular simulation requirement. In order to implement a priori abstractions, it is necessary to define abstractions that are valid for a significant set of simulation requirements. The importance of the concept of soundness is that it provides a way of considering model abstractions independent of a specific simulation requirement.

11.2 Future Directions

As with many research programs, this effort has raised questions as well as answered them. In particular, there are several ideas and concepts that warrant further demonstration as proof of concept demonstrations to demonstrate their validity. In this section, we summarize several directions in which this study should be continued to better understand and apply the concepts of model abstractions. Specific tasks include the following:

1. Formally define the abstraction taxonomy (possibly using the DEVS formalism).
2. Implement the RWWM, ALARM and/or ARES model abstractions and compare results of abstracted model and original model.
3. Demonstrate metamodeling as a surrogate abstraction technique using TERSM (or RWWM or ALARM). Take the metamodel developed by Zeimer et. al., and select fitting parameters. Modify the model accordingly. Execute test cases with original model, metamodel, and abstracted model to show variance under several scenario conditions.
4. Demonstrate use of qualitative models as surrogate techniques. Again using TERSM (or RWWM or ALARM), follow the approach used in Falkenhainer and Forbus for building a qualitative model from a quantitative model. Analyze the resulting qualitative model. Using this qualitative model, perform abstractions, then apply abstractions to main model.

Each of these research thrusts is described in more detail in a subsection below.

11.2.1 Formal Definition of Abstraction Techniques in the Taxonomy

Many of the concepts introduced in this report are defined informally and imprecisely. While an intuitive argument is made that they work in practice, and further research is recommended to demonstrate that more directly (see Section 11.2.2), a more scientific foundation for the use of model abstractions is required for a number of reasons:

- One of the goals of this research was to begin to build the knowledge necessary to allow models to be simplified without revalidation by validating the process by which they are simplified. Clearly the process cannot be validated without a formal definition of the process and the objects (models) to which the process is applied.
- The categorization of techniques in the taxonomy presented in Section 5 is based on our engineering judgment. A more precise definition of the abstraction techniques would allow new techniques to be unambiguously categorized, and would indicate whether the taxonomy is complete at a particular level.
- A precise formal description of techniques common to techniques derived from both the discrete event simulation domain and the model-based reasoning domain would facilitate the exchange of concepts and approaches.

The recommended research would involve using an existing formalism such as DEVS as the basis for defining abstraction techniques. Then specific criterion could be defined for other terms that have been introduced in this report, such as valid and sound abstractions.

11.2.2 Implementation/Evaluation of Abstractions

The discussion of actual abstractions in Sections 6, 7, and 8 identified a number of approaches, but does not include discussion of actual implementation of the abstractions. Implementation and evaluation is a logical next step. While any (or all) of the three models (RWWM, ALARM, or ARES) could be the focus of such a study, we recommend beginning with ALARM for the following reasons:

- Since ALARM is currently being used, the results could have immediate benefit outside Rome Laboratory.
- The results could be compared with other studies that have been conducted using ALARM (such as sensitivity analyses).
- The model is currently supported, reducing the chance of other software issues interfering with the analysis of abstractions.

The study should be conducted largely as a series of tests, in which the abstraction is applied to the baseline model, and then executed against a variety of test cases. The collected data should include the measures of effectiveness of the model, along with run-time execution and other performance measures. The general questions to be answered include the following:

- How do the MOEs of the abstracted model compare to those of the baseline model?
- Is the performance of the abstracted model significantly better than that of the baseline model?

Clearly the first question deals with whether the abstractions are valid, while the second question deals with whether the abstraction was useful.

ARES provides another interesting test case, particularly the Communications submodel and the Intelligence, Surveillance, and Reconaissance (ISR) submodel, since they can be used to illustrate the significance of model abstractions in evaluating model integration efforts. In both the Communications and ISR submodels, several levels of fidelity are already defined. The analysis in Sections 8.2.3 and 8.2.4 indicates the extent to which the lower fidelity versions of those submodels can be considered abstractions of the higher fidelity models. Analysis of implemented versions of those submodels can be used to determine the utility of that analysis.

11.2.3 Metamodel Surrogate Abstraction Demonstration

The purpose of this research task is demonstrate the concept of using metamodeling as a surrogate model abstraction technique. Surrogate abstraction techniques were identified in Section 10.3.3 as a technique for simplifying discrete event simulation models. By translating a model into a different (surrogate) form, it is easier to identify the types of simplifications that can be applied to the model. One such surrogate domain is that of metamodeling. The principle

of metamodeling is to create a mathematical expression of model outcomes based on analysis of the model input/output relationships. The value of the metamodel in this context is that it can isolate the relationship of input parameters to facilitate identification of candidate model simplifications.

The proposed research would build on the metamodeling work to demonstrate the application of metamodels as surrogate abstraction techniques. Specifically, we propose taking a model such as TERSM (for which metamodels have already been developed), using the metamodels to identify abstractions, implementing those abstractions as modifications to the original model code, then comparing the results of the original model, the metamodel, and the abstracted model.

The starting point for such a study would be the TERSM metamodels described in Section 10.2.3.3 and Appendix E. As discussed in Section 10.2.3.3, the metamodels suggest that he TERSM model ourtputs are less sensitive to altitude than to velocity. This hypothesis should be tested by implementing two abstractions: eleimination of altitude, and elimination of velocity. Abstractions are implemented by modifying the code of the original model. For example, a candidate abstraction which eliminates a parameter from consideration might be implemented by simply using a zero value for the input. In other cases, where a zero value has particular significance or cannot be set to zero (e.g., if the input parameter is a term in a divisor), the modification to the model may require additional code modification. Hypotheses based on other candidate abstractions that can be derived from a more detailed analysis of the TERSM metamodels should also be evaluated.

The final task of such a research effort is to compare the results of the original model, the metamodel, and the abstracted model. The objective of this analysis is to determine whether the abstracted model was a valid simplification of the original model, and whether the outputs predicted by the metamodel were indeed reflected in the abstracted model output.

11.2.4 Qualitative Surrogate Abstraction Demonstration

The project described in the preceding section will establish the use of metamodeling as a surrogate abstraction technique. The next research step should be to establish qualitative modeling as a surrogate domain. This relationship is not as clear, because (in the terminology of Section 10.3) the θ transformation is not nearly as well defined for qualitative models as for metamodels.

This study will involve several steps:

1. Begin by taking a simple discrete event model and derive an equivalent qualitative version of the model.
2. Then apply the various abstraction techniques to the qualitative version of the model, evaluating the effectiveness of the various techniques.

3. Apply the most effective techniques to the original baseline model.
4. Repeat the first step for a component of one of the models used in this study (TERSM, RWWM, ALARM, or ARES).

Each of these steps is discussed in more detail in a paragraph which follows:

Task 1—Derive a Qualitative Model from a Simple Discrete Event Model: Little work has been reported in the research literature on deriving qualitative models from discrete event models. Forbus and Falkenhainer (1992) describe an approach in which a qualitative model is used as the basis for explanations of the outcomes of discrete event simulation models, and they define an approach for establishing a mapping between states of the qualitative model and the states of the discrete event model. However, no explicit approach for deriving a comparable model is defined; one is constructed to match the physical process being modeled by the discrete event model.

We recommend a simple model be used for this study (rather than the models used in the current study) to minimize the complexity of the model transformation. Appendix F includes an anallysis of a qualitative version of a simple model of boiling water originally used in a study of multimodeling by Fishwick and Zeigler (1992). However, this analysis was not carried through to an actual qualitative model.

In order to ensure that the model is properly constructed, a means to execute the qualitative model is required. The most appropriate candidate is the QSIM system developed by Kuipers (1994). Kuipers acknowledges that building qualitative models with QSIM "is still something of an art", so we do not have a recommended methodology for accomplishing this first task.

Task 2—Apply/Evaluate Abstraction Techniques: The second task is to apply the techniques described in Section 4 to the qualitative model derived in Task 1. Not all techniques will be applicable, so part of the task will involve determining which techniques should be applied. For each technique that is implemented, the effect of the abstraction in terms of model outcomes and performance must be determined.

Task 3—Apply Abstractions to Original Model: Although difficult, the preceding two tasks have some precedent. This task involves the most innovative and challenging part of this suggested study. The use of qualitative models as a surrogate abstraction domain hinges on the ability to translate abstractions from the qualitative domain to the discrete event domain. In some cases, the mapping is fairly intuitive. For example, if a parameter can be identified as a fitting parameter (Weld 92) in the qualitative version of the model, then the mapping applied to the original discrete event model is elimination of that parameter as an exogenous input. Other qualitative abstractions may be more difficult to implement in the discrete event domain, such as state aggregations. Part of this task is to determine the transportability of such abstractions across domains, and to develop approaches to implementing such abstractions.

The implementation of an abstraction on the baseline discrete event model actually includes several steps:

- Modification of the model code, including debugging/verifying the model.
- Execution of the model against a set of test cases to determine the effect of the abstraction on model outcomes.
- Collection of data to assess the impact of the abstraction on resource requirements.

The end result of this task is a set of abstracted versions of the original model, which can then be selected for execution based on a specific simulation requirement.

Task 4—Repeat Task 1 for More Complex C^3I Model: Because of the research nature of this study, we recommended a simple model be the basis of the first three tasks. That leaves the question of whether the concept of qualitative models as a surrogate domain can be scaled up to models of the complexity of TERSM, RWWM, ALARM, ARES, and so on. The final step of this research thrust is to look at the applicability of this approach to a real C^3I model, such as those cited above. For the purposes of the research effort, the question can be addressed by building a qualitative version of such a model, i.e., repeating Task 1 for one of those four models. We suggest TERSM to provide a basis for comparison with other studies being sponsored by Rome Laboratory which are also using TERSM.

11.3 Conclusions

Model abstractions are an important tool in the design, development, and utilization of simulation models. While simulation practitioners have been applying abstraction techniques since the beginning of simulation program development, their potential remains underutilized because little has been done to build a body of scientific knowledge as to how to identify, apply, and utilize model abstractions to facilitate simulation.

This research effort is a first step in the process of providing some organization and foundation for understanding and exploiting model abstractions, particularly in the domain of complex discrete event C^3I models. Like most research programs, we have answered some questions and raised many more, and much work remains to be done to truly realize the advantages of model abstractions.

We conclude with a vision of where this research, along with other ongoing studies in the simulation field, could lead. If simulation models can be manipulated to meet individual simulation requirements, in terms of required data, appropriate level of fidelity, and so on, then a single multimodel could represent a variety of information about an entity and its associated processes. The phrase "single multimodel" is not a contradiction in terms; the multiple facets of the multimodel represent different ontologies or perspectives of the physical entity/process, but it is a single integrated representation of the knowledge of the behavior of that system, from which one can extract the relevant information at "run time" for efficient execution to meet a simulation requirement. The fact that a model represents the modelers' knowledge of the dynamic behavior of a system truly fuses the disciplines of behavioral knowledge representation

(from the perspective of the AI community) and simulation and modeling (which has traditionally been a separate community). Model bases can be repositories of knowledge of dynamic system behavior.

Object oriented simulation is an integral part of this vision. We have not explicitly addressed the role of object oriented approaches to simulation and system development in this report because the abstraction techniques that have been discussed are generally independent of the implementation techniques. Parameter elimination can be implemented in a traditional model as well as an object oriented model. An object oriented approach makes it generally easier to implement abstractions, just as it facilitates many other aspects of simulation model design, development, and implementation. The analysis of abstraction opportunities was not significantly different for ARES, which is being developed as an object oriented model, as for RWWM and ALARM, developed as "traditional" models.

Model abstractions are the keystone of this vision. With today's state of the art, we can build models of arbitrary complexity that represent every aspect imaginable of a real-world system. THe problem is the complexity of the model, and the fact that the resources required to build and execute such a model are generally well beyond what is either necessary or available to address a particular problem. For that reason, along with other problems in developing reusable software, it has traditionally been less expensive to develop new models to meet specific simulation requirements. With the technical advances in the area of model abstractions, in conjunction with other research and development activities in the areas of object oriented simulation and modeling, software reuse, and knowledge representation, simulation models can be developed and used much more efficiently and effectively.

12. References

Addanki, S., R. Cremonini, and J. Penberthy. 1991. Graphs of models. *Artificial Intelligence* 51:145-177.

Agrawal, S.. 1985. *Metamodeling: a study of approximations in queuing models.* MIT Press, Cambridge, Massachusetts.

Berleant, D., and Kuipers, B. 1992. Qualitative-numeric simulation with *Q3.* In *Recent Advances in Qualitative Physics*, Faltings and Struss, eds. MIT Press, Cambridge, Massachusetts.

Burton, R. 1994. Metamodeling: a state of the art review. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 237-244. Institute of Electrical and Electronics Engineering, San Francisco, California.

Caughlin, D. 1994. A metamodeling approach to model abstraction. In *Proceedings of the Fourth Dual Use and Technologies Conference*, 387-396. Institute of Electrical and Electronics Engineering, San Francisco, California.

Clancy, D., and Kuipers, B. 1993. Behavior abstraction for tractable simulation. In *Proceedings of the Seventh International Workshop on Qualitative Reasoning About Physical Systems (QR-93).*

Clancy, D., and B. Kuipers. 1994. Model decomposition and simulation. In *Working Papers of the Eighth International Workshop on Qualitative Reasoning About Physical Systems (QR-94).* Nara, Japan.

D'Ambrosio, B. 1986. *Qualitative process theory using linguistic variables.* Ph.D. Thesis. University of California, Berkeley, California.

de Kleer, J., and Brown, J. 1984. A qualitative physics based on confluences. *Artificial Intelligence* 24:7-83.

Dubois, D., and Prade, H. 1980. *Fuzzy sets and systems: theory and applications.* Academic Press, San Diego, California.

Falkenhainer, B. and K. Forbus. 1991. Compositional modeling: finding the right model for the job. *Artificial Intelligence* 51:95-143.

Fishwick, P. 1988. The role of process abstraction in simulation. *IEEE Transactions on Systems, Man, and Cybernetics* 18:18-39.

Fishwick, P. 1992. An integrated approach to system modeling using a synthesis of artificial intelligence, software engineering, and simulation methodologies. *ACM Transactions on Modeling and Computer Simulation* 2:307-330.

Fishwick, P. and B. Zeigler. 1992. A multimodel methodology for qualitative model engineering. *ACM Transactions on Modeling and Computer Simulation* 2:52-81.

Fishwick, P., N. Narayanan, J. Sticklen, and A. Bonarini. 1994. A multimodel approach to reasoning and simulation. *IEEE Transactions on Systems, Man, and Cybernetics* 24:1433-1449.

Forbus, K. 1984. Qualitative process theory. *Artificial Intelligence* 24:85-168.

Forbus, K., and Falkenhainer, B. 1992. Self-explanatory simulations. In *Recent Advances in Qualitative Physics*, Faltings and Struss, ed., MIT Press, Cambridge, Massachusetts.

Frantz, F. 1994. Reusing simulation software. In *Proceedings of the Fourth Dual-Use Technologies and Applications.* Institute of Electrical and Electronics Engineering, San Francisco, California.

Frantz, F. 1995. A taxonomy of model abstraction techniques. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 1413-1420. Institute of Electrical and Electronics Engineering, San Francisco, California.

*Frantz, F., and Brown, C. 1992. *Two way sensor interface.* Rome Laboratory Technical Report No. RL-TR-92-234, Rome Laboratory, Griffiss Air Force Base, New York.

*Frantz, F., Gillmore, D., Brown, M., Foreman, S., and Pelkey, R. 1990. *Joint C3 simulation.* Rome Laboratory Technical Report No. RL-TR-90-262, Rome Laboratory, Griffiss Air Force Base, New York.

General Research Corporation (GRC) and Computer Sciences Corporation (CSC), 23 October 1995. *Requirements Document for the Advanced Regional Exploratory System (ARES).* Document Number ARES-95/07/21-PEZ-001-v1.2-CD.

General Research Corporation (GRC) and Computer Sciences Corporation (CSC), 31 October 1995. *Software Design Document for the Advanced Regional Exploratory System (ARES).* Document Number ARES-95/10/31-CRR-001-v0.1-CD.

Ho, Y. 1989. *IEEE Proceedings on DEDS (Discrete Event Dynamic Systems)*, ed. Yu-chi Ho. Institute of Electrical and Electronics Engineering, San Francisco, California.

Ijiri, Y. 1971. Fundamental queries in aggregation theory. *Journal of the American Statistical Association* 66.

Iwasaki, Y., and Simon, H. 1986. Causality in device behavior. *Artificial Intelligence* 29.

Iwasaki, Y. 1988. Causal ordering in a mixed structure. In *Proceedings of the Sixth National Conference on Artificial Intelligence.* American Association of Artificial Intelligence/ MIT Press, Cambridge, Massachusetts.

Iwasaki, Y. 1992. Reasoning with multiple abstraction models. In *Recent Advances in Qualitative Physics*, Faltings and Struss, ed., MIT Press, Cambridge, Massachusetts.

Jefferson, D. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7:404-425.

Karp, P., and Freidland, P. 1989. Coordinating the use of qualitative and quantitative knowledge in declarative device modeling. In *Artificial Intelligence, Simulation, and Modeling,* Widman, Loparo, and Nielsen, eds., Wiley & Sons, New York, New York.

Kuipers, B. 1986. Qualitative simulation. *Artificial Intelligence* 29:289-338.

Kuipers, B. 1994. *Qualitative reasoning: modeling and simulation with incomplete knowledge.* MIT Press, Cambridge, Massachusetts.

Miller, D., J. Firby, P. Fishwick, D. Franke, and J. Rothenberg. 1992. AI: what simulationists need to know. *ACM Transactions on Modeling and Computer Simulation* 2:269-284.

Murthy, S. 1988. Qualitative reasoning at multiple resolutions. In *Proceedings of the Sixth National Conference on Artificial Intelligence.* American Association of Artificial Intelligence/ MIT Press, Cambridge, Massachusetts.

*RL TR-92-234 and RL-TR-90-262 are both Distribution limited to DOD and DOD contractors only; critical technology.

Nayak, P. 1992. Causal approximations. In *Proceedings of the Tenth National Conference on Artificial Intelligence,* 703-709. American Association of Artificial Intelligence/ MIT Press, Cambridge, Massachusetts.

Radiya, A. and R. Sargent, 1994. A logic-based foundation of discrete event modeling and simulation. *ACM Transactions on Modeling and Computer Simulation* 4:3-51.

Rickel, J., and P. Porter. 1994. Automated modeling for answering prediction questions: selecting the time scale and system boundary. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. American Association of Artificial Intelligence/ MIT Press, Cambridge, Massachusetts.

Science Applications International Corporation (SAIC). *Operational concept document (analyst's manual) for the Advanced Low Altitude Radar Model (ALARM 3.1).* Survivability/Vulnerability Information Analysis Center. Report Number 95-249, DCN 1123. June, 1995.

Science Applications International Corporation (SAIC). *Software user's manual for the Advanced Low Altitude Radar Model (ALARM 3.1).* Survivability/Vulnerability Information Analysis Center. Report Number 95-250, DCN 1124. June, 1995.

Shen, Q., and Leitch, R. 1992. Combining qualitative simulation and fuzzy sets. In *Recent Advances in Qualitative Physics*, Faltings and Struss, ed., MIT Press, Cambridge, Massachusetts.

Simon, H., and Ando, A. 1961. Aggregation of variables in dynamic systems. *Econometrica* 29.

Weld, D., and Addanki, S. 1992. Query-directed approximation. In *Recent Advances in Qualitative Physics*, Faltings and Struss, ed., MIT Press, Cambridge, Massachusetts.

Weld, D. 1990. Approximation reformulations. In *Proceedings of the Eighth National Conference on Artificial Intelligence,* 407-412. American Association of Artificial Intelligence/ MIT Press, Cambridge, Massachusetts.

Weld, D. 1986. The use of aggregation in causal simulation. *Artificial Intelligence* 30:1-34.

Weld, D. 1992. Reasoning about model accuracy. *Artificial Intelligence* 56:255-300.

Widman, L. 1989. Semi-quantitative "close-enough" systems dynamics models: an alternative to qualitative simulation. In *Artificial Intelligence, Simulation, and Modeling,* Widman, Loparo, and Nielsen, eds., Wiley & Sons, New York, New York.

Williams, B. 1986. Doing time: putting qualitative reasoning on firmer ground. In *Proceedings of the Fourth National Conference on Artificial Intelligence,* 105-112. American Association of Artificial Intelligence/ MIT Press, Cambridge, Massachusetts.

Zeigler, B., and Weinberg, R. 1970. System theoretic analysis of models: computer simulation of a living cell. *Journal of Theoretical Biology*, 29.

Zeigler, B. 1990. *Object oriented simulation with hierarchical, modular models: intelligent agents and endomorphicsystems*. Academic Press, San Diego, California.

Zeigler, B. 1976. *Theory of modeling and simulation.* New York, New York: Wiley and Sons.

Zeigler, B. 1984. *Multifaceted modeling and discrete event simulation.* San Diego, California: Academic Press.

Zeimer, M., Tew, J., Sargent, R., and Sisti, A. 1993. Metamodeling procedures for air engagement simulation models.

A. Synopsis of DEVS and SES

The purpose of this appendix is to provide a synopsis of the concepts of the Discrete Event System Specification (DEVS) formalism developed by Zeigler. This appendix is drawn directly from Zeigler's texts, particular (Zeigler, 1984), which contains the most detailed discussion of the DEVS concepts. This synposis was originally written to assist the project team in identifying the key concepts of the DEVS formalism that related to the issue of model abstraction. Since it provides a concise summary of key definitions and concepts, it is included as an appendix to this report.

This appendix is divided into three parts. We begin with a description of the DEVS formalism. The concept and examples of morphisms are then presented in Section A.2. We then present a description of the SES.

A.1 Discrete Event System Specification (DEVS) Formalism

A **formalism** expresses the structure of a model in a mathematical language. A discrete event formalism focuses on the changes of variable values and generates time segments that are piecewise constant. An event is thus defined as an instantaneous change in a variable value. Note that in a formalism the time intervals between event occurrences may be variable.

We begin with a brief summary of the formalism developed by Zeigler. Zeigler has developed an extensive theoretical foundation called the Discrete Event Simulation (DEVS) Formalism. In this section, we summarize briefly the salient points of this theoretical foundation, particularly as it applies to our investigation of model abstraction approaches. This summary is taken primarily from (Zeigler 1984) and Zeigler 1990), which, along with (Zeigler 1976), are the texts that provide the most detailed exposition of this theoretical foundation. There are numerous papers that have also been published on this topic, many of which provide examples in which the theory has been applied to a modeling problem. Bold type is used as in the texts to denote definitions.

We begin with some definitions that characterize formalisms in general, before discussing the specific DEVS formalism. We start with the definition of a **representation scheme**, which is a means of representing reality in computerized form. Each representation adhering to the pattern laid down by the scheme consists of four different features or **slots**:

- Operations;
- Questions;
- Designation of what is represented; and
- Correspondence with what is represented.

Formalisms in simulation are best regarded as set-theoretic shorthand for specifying mathematical dynamic systems:

- that are often associated with particular simulation languages;
- that have an independent conceptual existence; and
- for which some are more "natural" than others.

The same system may have a variety of related models expressed in different formalisms.

Zeigler's theoretic foundation for discrete event models is based on a more general formal description of a system. Specifically, he defines a Discrete Event System (DEVS) formalism, which provides the means of specifying a mathematical object called a system. A system has:

- A time base;
- Inputs and outputs;
- States; and
- Functions for determining next states and outputs given current states and inputs.

Parameters of discrete systems parallel parameters of continuous systems, e.g., inputs in discrete systems occur at arbitrarily spaced moments, while inputs in continuous systems are piecewise continuous functions of time.

The motivation for defining the DEVS is that it provides insights by the simple way it characterizes how discrete event simulation languages specify discrete event system parameters. Furthermore, it also provides a formal representation of discrete event systems capable of mathematical manipulation just as differential equations serve this role for continuous systems. The DEVS formalism is most closely related to formalisms emerging under the framework of Generalized Semi-Markov Processes (GSMP), including Petri Nets.

The DEVS formalism is based on the relationships between the real-world system being modeled, the model, and the simulator. On this base, Zeigler adds the notion of an **experimental frame**, which captures how the modelers' objectives impact on model construction, experimentation, and validation. An experimental frame is paired with each model to form coupled model objects that are amenable to formal analysis and manipulation.

Basic models (from which larger models are composed) are defined in DEVS by the structure:

$$M = (X, S, Y, d_{int}, d_{ext}, l, ta)$$

where

X	=	Set of external input event types.
S	=	Sequential state set.
Y	=	Set of external event types generated as output.
d_{int}	=	Internal transition function, dictating state transitions due to internal events.
d_{ext}	=	External transition function, dictating state transitions due to external events.

l = Output function generating external events at the output.
ta = Time advance function.

Typically:

- External transition on input port is in form: when receive x on input port P .....
- Internal transition functions are specified in process form with phases and transitions
- Output functions are in the form: Send y to output port P

Input and output sets X and Y consist of pairs of the form x = (p,v), signifying that an input value v has arrived on port p.

Multi-component models are implemented as coupled models. A coupled model contains the following information:

- the set of **components**;
- the **influencees** of each component;
- the set of **input ports** through which external events are received;
- the set of **output ports** through which external events are sent;
- The **coupling specification**, consisting of:
  - the external input coupling - directs inputs received at input ports of the coupled model to input port(s) of one or more component;
  - the external output coupling - connects output port(s) of component(s) to output port(s) of coupled model to be transmitted externally;
  - the internal coupling - connects output ports of components to input ports of other components; and
- the **select function** - chooses which component is allowed to carry out its next event.

State changes of components can then be expressed in the DEVS formalism, starting at elapsed time "t" when each component "i" has been in its current state s_i for an elapsed time e_i.

A DEVS simulation environment has been implemented, based on these concepts. It is a general purpose environment for constructing hierarchical discrete event models. It is written in DOS based PC-Scheme language running under a Scheme interpreter for the Texas Instruments Explorer. Developed models are readily transportable to distributed simulation systems. Finally, it provides a convenient basis for development of learning or evolutionary models since structure descriptions are accessible to run-time modification.

All classes in DEVS-Scheme are subclasses of the universal class `entity`. Main classes are:

- Models
  - atomic-models
  - coupled models
- Processors
  - simulators
  - coordinators
  - root coordinators

`Models` and `processors` serve to provide slots needed by their specializations. For example, the class `models` has instance variables `processor` (which records responsible processor) and `parent` (which records the parent coupled model, if any). The class `models` has two sub-classes, `atomic models` and `coupled models`. The class `atomic models` realizes the atomic level of the formalism. An `atomic model` has instance variables such as `int-transfn`, which specifies the model's internal transfer function, and `ext-transfn`, which specifies the model's external transfer function. The class `coupled models` embodies the hierarchical model composition constructs, defining: the children components of the model; sibling models that receive output generated by the model; and subordinate components that receive external events generated by the model.

Simulators, coordinators, and **root coordinators** are specialization classes of processors. Simulators carry out instructions implicit in atomic models to generate their behavior. Coordinators carry out instructions implicit in coupled models to generate their behavior. Simulators and coordinators handle models on a one to one basis. The root coordinator manages the overall simulation.

Processors handle the message traffic. Messages contain:

- source - of origin;
- time - local or global time stamp; and
- content - port designation and a value.

A.2 Levels, Morphisms, and Associations

The concept of morphisms and associations provides a mechanism for specifying systems at various levels of resolution, and establishing relations between specifications. Morphisms defined at a single level establish the equivalence of system structure and/or behavior at that particular level. Associations and morphisms between levels provide a means to generate

behavior from structure (downward direction) and recover structure from behavior (upward direction). The key to the morphisms and associations is the levels of system specification.

A.2.1 Levels of Resolution

There are six levels of resolution defined by Zeigler, numbered from 0 to 5. Each level is defined briefly below.

Level 0. Observation Frame.

An **observation frame** is a structure

$$O = \langle T, X, Y \rangle$$

where

T is a set, the **time base**

X is a set, the **input value** set

Y is a set, the **output value** set

Thus an observation frame defines a behavior that over the time frame represented by T, the inputs X are input to the system, generating outputs Y.

Level 1. I/O Relation Observation.

The I/O Relation Observation provides greater resolution by including the I/O relations between the input set and the output set. Specifically, an input/output relation observation is a structure:

$$IORO = \langle T, X, \Omega, Y, R \rangle$$

where

$\langle T, X, Y \rangle$ is an observation frame

Ω is a set, the input segment set

R is a relation, the **I/O Relation**

subject to the constraints:

$$\Omega \subset (X, T)$$

$$R \subset (\Omega \times (Y, T))$$

such that if $(\omega, \rho) \subset R$ the $\text{dom}(\omega) = \text{dom}(\rho)$. This constraint ensures that both segments of the I/O pair are observed over the same time interval.

Level 2. I/O Function Observation.

The I/O Function increases the level of specification by replacing the I/O relation with a more specific set of functions. An I/O Function Observation is a structure

$$IOFO = \langle T, X, \Omega, Y, F \rangle$$

where

$\langle T, X, Y \rangle$ is an observation frame

Ω is a set, the input segment set

F is a set, the set of **I/O Functions**

subject to the constraints

$f \in F$ implies $f \subset \Omega \times (Y, T)$

f is a function such that $\text{dom}(f(\omega)) = \text{dom}(\omega)$

Level 3. I/O System.

The I/O System level introduces the notion of system states and memory, thus providing more detail in the system specification. An I/O System is a structure

$$S = \langle T, X, \Omega, Y, \delta, \lambda \rangle$$

where

T, X, Ω, Y are as in Level 2

δ is a function, the transition function

λ is a function, the output function

Q is the set of internal states that represent the memory of the system. The state transition function is a map δ: $Q \times \Omega \rightarrow Q$. If a system is in state q at time t_i, and an input segment is applied over the interval $[t_i, t_f]$, $\delta(q,\omega)$ is the state of the system at time t_f. The output function is a map $\lambda: Q \rightarrow Y$, interpreted as follows: when a system is in state q, the output of the system is $\lambda(q)$.

Level 4. Structured Systems.

A system specification at this level is the same as that at Level 3 except that each of the sets and functions is structured. The term structured is used with same meaning as in structured set; a structured system can be represented as the crossproducts of more elementary sets and functions.

Level 5. Coupling of Systems.

A coupling of systems provides the highest level of specification by allowing a system to be defined in terms of other systems. Specifically, the specification designates a set of components and a coupling scheme that defines the relationship of components.

A.2.2 Morphisms

Morphisms define mappings between two systems that preserve the features at a particular level. Morphisms are defined as onto relations from "larger" systems (i.e., those with more states) to "smaller" systems. If there is a morphism from the smaller to the larger system (i.e., if the morphism is 1:1 and onto), the morphism is called an **isomorphism**. For example, two systems are isomorphic at the observation frame level if over the same time period, the same inputs and outputs are observed.

Morphisms and associations provide the mechanisms to relate system specifications at different levels. In the downward direction, behavior specifications are generated from structural specifications; in the opposite direction, the structure can be derived from behavior.

A.3 System Entity Structure

The concept of the System Entity Structure (SES) builds on several key concepts, discussed in the following sections. We begin with the concept of aggregation (Section A.3.1), followed by concepts relating to coupling multiple models (Section A.3.2). We conclude this section with a discussion of the framework implied by the SES.

A.3.1 Aggregation Concepts

Models may be constructed at different levels of aggregation. More aggregation equates to less resolution, while more disaggregation equates to more resolution. The opposite of abstraction is also referred to as **concretization:** making the abstract more concrete[1]. A simulation environment can support construction of aggregation related models by facilitating elaboration and simplification of models. Relationships among collections of models should form part of the knowledge base of the environment. The experimental frame concept provides a basis for rigorously characterizing behavioral regions where validity is expected to hold.

There is a hierarchy of levels in which simulation models can be constructed, ranging from purely behavioral to strongly structural. Simulation models are usually at a higher structural level; curve fitting is example of behavioral. Conventional simulation supports only a single level at which change occurs, that of change in model descriptive variables. A new paradigm is needed to avoid forcing structural changes down to the same level as behavioral ones.

[1]This definition equates to the notion of refinement, although refinement has the connotation of the inverse of approximation rather than abstraction. Another term is disaggregation, although again if disaggregation is assumed to be decomposition, it is not exactly equivalent to concretization.

A.3.2 Concepts of Coupling Models

This paradigm is based on notions of coupling and modularity. **Coupling** is defined as the creation of a model AB in the model base by specifying how the inputs and outputs of existing models A and B are connected to each other. AB is called a **coupled model**. **Modularity** is defined as the description of a model in such a way that it has recognized input and output ports through which all interaction with the outside world is mediated. A coupling specification has three parts:

- external input coupling;
- external output coupling; and
- internal coupling.

Independent testability is an important result of modularity of models. Experimental frames specify input, control, and output variables and constraints desired for test.

A **hierarchical model** is inductively defined such that:

- an atomic model is a hierarchical model;
- a coupled model whose components are hierarchical models is a hierarchical model; and
- nothing else is a hierarchical model.

The structure of a hierarchical model is exhibited in a composition tree. Child models that are atomic models become leaves on the composition tree.

A.3.3 SES Framework

The **system entity structure** (SES) provides the framework for structuring a model to realize these concepts. In it an entity may have several decompositions called **aspects**. Each such aspect carries with it the coupling specifications needed to construct the parent by coupling together the children of that aspect. The SES directs the synthesis of models from components in the model base. It is a knowledge representation scheme that combines the decomposition, taxonomic, and coupling relationships.

Associated with an SES is a **model base** containing models that may be expressed in any of the dynamic formalisms mentioned earlier. The **entities** of the SES are conceptual components of reality for which models may reside in the model base. Another useful notion is the organizational concept of **specialization**. Specialization is a means of representing the possible variants or forms that an entity might assume. It is similar to an aspect decomposition.

A **pruned entity structure** (PES) is defined as an entity structure where **pruning** has eliminated undesirable alternates. Not all choices may be selected independently; "rules may be associated with the structure" so that choices taken in one branch path affect options in other

paths. Pruned entity structures are stored along with the SES in files forming the **entity structure base**. A **transform** is employed to traverse the pruned entity structure. As it does so, it searches the model base for a model of the current entity. The ability to examine pruned entity files is significant for reusability.

A.4 References

Zeigler, B. 1976. *Theory of modeling and simulation.* New York, New York: Wiley and Sons.

Zeigler, B. 1984. *Multifaceted modeling and discrete event simulation.* San Diego, California: Academic Press.

Zeigler, B. 1990. *Object oriented simulation with hierarchical, modular models: intelligent agents and endomorphicsystems*. Academic Press, San Diego, California.

B. RWWM Design Documentation

This appendix contains the design information that was extracted from the RWWM model code as part of the analysis of abstraction techniques. This information was derived by examining the code and in-line documentation. The appendix includes the following tables:

There is also a set of figures showing the calling structure of subroutines in the model. This information is useful in determining the propagation of effects of input variables.

Table B-1, RWWM Event Types and Processing

RWWM EVENT TYPES AND PROCESSING		
Event Type #	**Description**	**Model Subroutines Executed**
1	Ground-Air Intercept	ACPOS, WESDEL[1], AFPOUT
2	Air-Ground Detonation	ACPOS, CPKS[1], SITDEL[1], AFPOUT
3	Start Evasive Maneuver	MANPTS[1], AFPOUT
4	Stop Evasive Maneuver	MANPTS[1], AFPOUT
5	End of Attack	THISPT, AFPOUT, BEAR
6	Reactive Point	THISPT, AFPOUT, BEAR
7	Initial Activation	ACPOS, EOBGEN, APR38, APROUT
8	Beginning of Update Cycle End of Update Cycle	ACPOS, EOBGEN, APR38, APROUT, BEAR, THISPT, FPOUT
9	Scan Table Update	ACPOS, EOBGEN, APR38, APROUT
10	Scan Table Update	ACPOS, EOBGEN, APR38, APROUT
11	Scan Table Update	ACPOS, EOBGEN, APR38, APROUT
12	Scan Table Update	ACPOS, EOBGEN, APR38, APROUT
13	Activate/Deactivate	SITES
14	Change Track	SITES
15	Pre-Planned Breakpoint	TYPE15, AFPOUT
16	Pre-IP Point	ATTPTS[1], AFPOUT
17	IP Point	ATTPTS[1], AFPOUT
18	Pull Up	ATTPTS[1], AFPOUT
19	Roll In Point	ATTPTS[1], AFPOUT
20	Roll Out Point	ATTPTS[1], AFPOUT
21	Release Point	ATTPTS[1], AFPOUT
22	Roll Off Point	ATTPTS[1], AFPOUT
23	End of Turn to Head Back Toward Course	TOCORS[1], AFPOUT
24	Start Turn to On Course On Time Point	TOCORS[1], AFPOUT
25	On Course On Time Point	TOCORS[1], AFPOUT
26	Return to Base Point	WESDEL[1], AFPOUT
27	Start of Attack by Weasel Against Ground Site	[2]
28	Weapon Airborne Against Site Delete	DVLDWEP
29	Top of Pop-Up Maneuver	ACPOS, plus set ALTCHG, TYME, CLMRATE

[1]Via call to WWDRIV.

[2]The review of event processing indicates that event type 27 is ignored in the model.

Table B-2, RWWM Input Parameters

RWWM INPUT PARAMETERS		
Variable Name	**Description**	**Data Representation**
RADSYS	Radar system description	Array indexed by radar system number, parameter numbers as follows: 1: Radar system identified 2: Priority 3: Critical range 4: Target track db threshold 5: Missile guidance db threshold 6: Launch db threshold 7: Launch range threshold
PT	Scan table processing time	Array indexed by scan table number 1..5, 1..EMAX
AS	Air speed for various modes of operation	Array indexed by mode of operation
TRNRAD	Turn radius	Scalar
TRNRATE	Turn rate	Scalar
PULRATE	Pull up rate	Scalar
PPFP	Pre-planned flight path (the initial set of break points scheduled for each Wild Weasel aircraft)	Array indexed by breakpoint number, break point parameter as listed below, and Wild Weasel number 1: X 2: Y 3: Altitude 4: Time 5: True heading 6: RTB:return to base heading 7: Bingo level
WWMTYP	Mission type	Array indexed by Wild Weasel number
TOT	Time over target	Array indexed by Wild Weasel number
USED	Fuel used	Array indexed by Wild Weasel number
FLIGHT	Pairing of Wild Weasels into flights	Array indexed by Wild Weasel number and 1..2
FLTNUM	Identifier of paired Weasel flights	Array indexed by Wild Weasel number
STORES	Weapons carried by each aircraft	Array indexed by Wild Weasel number and 1..9 (an aircraft can carry up to nine weapons)[3]
PST	Threat parameters associated with a site	Array indexed by Wild Weasel number, mission site number, and parameter number as follows: 1: Flag indicating pre-briefed emitter (PBE) 2: Threat of emitter to strike force 3: Flag indicating whether Weasel pre-planned flight path is within critical range of site
PKWEP	Weapons Pk data	Array indexed by type of site and by type of weapon
WEPREL	Weapons release range	Array indexed by weapon class

[3]There are actually nine weapons stations, and multiple AGM-65s can be loaded into a single station; for all other weapon types, only one weapon can be loaded per station.

RWWM INPUT PARAMETERS		
Variable Name	**Description**	**Data Representation**
WEPRNG	Weapons normal intercept point range	Array indexed by weapon class
WEPVAL	Weapon's velocity	Array indexed by weapon class
POSCHG	Weapons delivery recovery maneuver data	Array indexed by weapon type and index 1..6 for three sets of (angle, distance) coordinates
S	Radar site parameters	Array indexed by mission site number, parameter number as follows: 1: System type 2: Scan table [1..5] 3: Radar site X coordinate 4: Radar site Y coordinate 5: Radar site Z coordinate 6: Track radar emitter signal strength 7: Active flag [0,1] 8: IPBE 9: Missile guidance radar emitter signal strength
RADXYZ	Radar site coordinates	Array indexed by mission site number, 1..3 (X, Y, Z)
TRACK	Track parameter flag	Array indexed by Wild Weasel number, radar site number

Table B-3, Summary of Exogenous Input Utilization

SUMMARY OF EXOGENOUS INPUT UTILIZATION		
Input Variable	**Utilization**	**Subroutines Ref. In**
RADSYS(r,1): Radar system id	Key value for radar system parameter record	APRPRI, MANPTS, NGAGED, RNGSET, SAM, TRACK
RADSYS(r,2): Priority	Used to prioritize emitters in the site EOB (added to a priority based on emitter type)	APRPRI
RADSYS(r,3): Critical Range	Used to gate out priority assignments for emitters that out of range Defines end of maneuver point Used to determine whether a Wild Weasel can be attacked by a site	APRPRI, MANPTS, NGAGED, RNGSET
RADSYS(r,4): Target Track db Threshold	Used to determine TRACK parameter (which is used to determine target priority and whether Weasel is being engaged)	TRACK
RADSYS(r,5): Missile Guidance db Threshold	Used to determine TRACK parameter (which issued to determine target priority and whether Weasel is being engaged)	TRACK
RADSYS(r,6): Launch db threshold	Determines SAM site mode of operations (which is used to determine target priority and whether Weasel is being engaged)	SAM
RADSYS(r,7): Launch range threshold	Determines SAM site mode of operations (which is used to determine target priority and whether Weasel is being engaged)	SAM
PT(i,j)	Scan table processing time; used to determine the time until the next event is scheduled	ST5PRT, TIMING
AS (I)	Airspeed for each of four operating modes (gather, attack, support, evade) Used to determine weapons release points, weapons release times, and pull up times for various weapons Used to determine whether an aircraft can get back on course on time Min and max airspeeds used to interpolate fuel consumption rate	AGM88, ATTACK, AVOID, BESTHED, EVADE, FUELUP, ONTIME, TOCORS, TYPE15
TRNRAD	Used to calculate aircraft position in turns (turns are defined in terms of radius and angular rate) Used to compute turn types and stop turn points Used to determine the point where the Wild Weasel can get back on course on time	ACPOS, CENTER, GETBCK, MANPTS, REACT, TRNTYPE
TRNRATE	Used to calculate aircraft position in turns (turns are defined in terms of radius and angular rate) Used to determine the event time of a maneuver event (e.g. next break point, stop turn, roll out, etc.)	ACPOS, AGM45, AGM65, GETBCK, MANPTS, REACT, RECOVR, TYPE15
PULRATE	Used to determine launch angle and location for AGM45 release[4]	EX45

[4]The launch angle is then passed to AR45PK to calculate the Pk of the target, but our version of this subroutine is only a stub.

SUMMARY OF EXOGENOUS INPUT UTILIZATION		
Input Variable	**Utilization**	**Subroutines Ref. In**
PPFP(p,1,w): X PPFP(p,2,w): Y PPFP(p,3,w): Altitude PPFP(p,5,w): Heading	Used to calculate actual flight path of the Wild Weasel, either as regular aircraft location updates as it follows its path or after performing an attack or evasive maneuver	AVOID, BESTHED, GETBCK, MIDHED[5], ONTIME, RECOVR, TYPE15[6]
PPFP(p, 4, w): Time	Used to determine where and when aircraft can get back on pre-planned flight path	AVOID, BESTHED, FUELUP, GETBCK, ONTIME, RECOVR, TYPE15
PPFP(p,6,w): RTB heading	Used to set Return to Base heading when aircraft completes mission	RTB
PPFP(p,7,w): Bingo level	Used to determine when aircraft must break off mission due to low fuel	FUELUP
WWMTYP	Used to determine whether aircraft should return to base	STATUS
TOT	Used to determine whether aircraft should return to base because too late to attack target	STATUS
USED	Used to limit amount of fuel available to aircraft	FUELUP
FLIGHT(w,1): Associated Wild Weasel FLIGHT(w,2): Wingman flag	Used to reference wingman of lead aircraft to update wingman's state, change aircraft mode	ACPOS, AGM45, AGM65, AGM88, ATSTAT, ATTCHG, BEAR, CPKS, DELPTS, DETIME, EVADE, FUELUP, GETBCK, GETHOM, IPNOW, LETHEX, MANPTS, PIPNOW, PREATT, PULUP, RELNOW, ROLIN, ROLOFF, ROLOUT, RTB, SITES, STATUS, SUPPRT, THISPT, TOCORS, TYPE15, WEAPON
FLTNUM	Used only to designate output files	AFPOUT, APROUT
STORES	Used to keep track of what weapons are available on each aircraft, what weapon will be used for an attack, when aircraft must return to base if all weapons expended, and additional fuel if aircraft so configured	CAPABL, FUELUP, PREATT, SUPPRT, WEPDEL, WINCHR
PST	Used to calculate attack and defensive priorities	ATTREG, CPCOMP, PRICOMP
PKWEP	Used to calculate probability of kill of a weapon against a ground site target	CPKS
WEPREL	Used to determine weapon attack flight path	ATTACK, ESTREL
WEPRNG	Used to determine weapon attack flight path	ATTACK, ESTREL
WEPVEL	Used to determine weapon detonation time	DETIME
POSCHG	Used to determine aircraft flight path after weapons release	RECOVR
S(i,1): System type	Not used, only read in and written out in EOB file	
S(i,2): Scan table #		
S(i,3): Site X coord.	Used to compute the range and relative azimuth from the Wild Weasel location to radar site	A65PRI, ARMPK, CPCOMP, DETIME, UPDATE

[5]X and Y coordinates only.

[6]In RELNOW and TOCORS, break point data is extracted from PPFP for further processing.

SUMMARY OF EXOGENOUS INPUT UTILIZATION		
Input Variable	**Utilization**	**Subroutines Ref. In**
S(i,4): Site Y coord.	Same as above	Same as above
S(i,5): Site Z coord.	Used to compute the range from the Wild Weasel location to radar site Used to determine AGM-65 weapon release Z coordinate Used to determine altitude from bomb release point to target to compute bomb descent time	A65PRI, AGM65, ARMPK, CPCOMP, DETIME, UPDATE
S(i,6): Track radar emitter signal strength	Used to determine whether an emission has sufficient strength to be detected	EOBGEN
S(i,7): Active flag	Indicates whether emitter is emitting	EOBGEN
S(i,8): IBPE	Not used	
S(i,9): Non-track radar emitter signal strength	Used to determine whether an emission has sufficient strength to be detected	EOBGEN
TRACK	Used to retrieve proper emitter strength when building the EOB	EOBGEN

Table B-4, RWWM Modules

LIST OF REACTIVE WILD WEASEL MODEL (RWWM) MODULES		
Name	**Function**	**Called by**
A65PRI	Defensive priority for AGM-65 release point	EST65, RECOVR
ACPOS	Aircraft position	LETHEX
AFPOUT	Create Flight Path Output File	LETHEX
AGM45	Compute AGM45 attack point coords for site attacked	ATTACK
AGM65	Compute AGM65 attack point coords for site attacked	ATTACK
AGM88	Compute AGM88/78 attack point coords for site attacked	ATTACK
APR38	Develops Electronic Order of Battle	LETHEX
APROUT	Outputs EOB into files	LETHEX
APRPRI	Compute priority of each emitter in EOB for APR38 process	PROCES
AR45PK	Compute P_K ellipses for AGM-45 launch	EX45
ARMPK	Outputs P_K	
ATSTAT	Check attack status for discontinued site or A/C change	BEAR
ATTACK	Select attack point	PREATT
ATTCHG	Delete events of changed attack	ATSTAT, EVADE, PREATT
ATTPTS	Call for current attack point	WWDRIV
ATTREG	Compute defensive priority of sector between WW & attack point	CPCOMP
AVOID	Direct A/C back onto planned path after deviation	BEAR
AZSET	Set AZVAL to 3, 6, 8, or 10 depending on azimuth angle	A65PRI, DEFSECT, PRICOMP
BEAR	Check WW condition code, calls appropriate subroutine	CPRIOR, LETHEX, TYP15
BESTHED	Find best (least danger) heading to get back on course	AVOID
CAPABL	Weapon availability for WW and capability against threat	WEAPON
CENTER	Center of turn for given radius	AGM45, AGM65, GETBACK, REACT, RECOVR, TYP15
CPCOMP	Cumulative priority for site based on weapon release point	CPRIOR
CPKS	Cumulative probability of kill of a site	WWDRIV
CPRIOR	Attack, defensive, and cumulative priorities of site	BEAR, EOBCHG
DEFSECT	Compute defensive priority of sector ahead of WW	AVOID, GETHOME
DELETE	Delete event from event list	ATTCHG, BEAR, DELPTS, LETHEX, SITDEL, WESDEL
DELPTS	Delete returun to course events from event list for WW changing mode	AGM45, AGM65, AGM88, EVADE, GETHOME
DETIME	Calculate weapon detonation time, add to event list	RELNOW
DISTICK	Calculate 3D distance between two objects	A65PRI, AVOID, BESTHED, CPCOMP, DEFSECT, DETIME, EST65, GETBACK, ONTIME, RECOVR
DSIGN8	Designate new operator highest priority threat for APR38	ATTCHG, PREATT
EOBCHG	Detect changes to EOB	BEAR
EOBGEN	Generate I/P files for APR38	LETHEX
EST45	Estimate of release point for AGM-45 missile	ESTREL
EST65	Estimates of release points for AGM-65 missile	ESTREL
ESTREL	Estimates of release points	CPCOMP
EVADE	Initiate evasive maneuver for WW being engaged	GATHER, ATSTAT, SUPPRT
EX45	Maneuver WW to line up for AGM-45 attack	PULUP

LIST OF REACTIVE WILD WEASEL MODEL (RWWM) MODULES		
Name	**Function**	**Called by**
FUELUP	Increment fuel consumption & sets bingo flag	BEAR
GATHER	Determine site to be attacked	BEAR
GETBCK	Compute turns to get back on course and time	AVOID
GETHOME	Heading for return to base via region with lowest defensive value	AVOID
HEAD	Calculate heading from point A to boint B	
INPROS	Select and updates scan table at start of scan cycle	APR38
INSERT	Insert an event into the event list	AGM45, AGM65, AGM88, BEAR, DETIME, EVADE, GETBACK, MANPTS, PULUP, REACT, RECOVR,RELNOW, RTB, TIMING, TOCARS, TYP15, WESDEL, WWDRIV
INTEG	Main program for APR38 stand-aloneoperation	
INTIAL	Read required input files	LETHEX
IPNOW	Initialize WW to IP for attack against site	ATTPTS
LETHEX	Exec for RWW integrated and stand-alone	INTEG
LNCRLE	Compute intersect of line & circle	ESTREL, AGM88, MANPTS
MANPTS	Compute turn to put attacking site @ 10 or 2 o'clock	WWDRIV
MIDHED	Compute average of two headings	AVOID, ONTIME
MOVAXS	Moves and rotates coordinate system	
MTSET	Sets the MODVAL and TRKVAL parameters	DEFSECT, PRICOMP
NGAGED	Scans EOB of an aircraft to see if it is being engaged	ATSTAT, GATHER, SUPPRT
OLDSIT	Processes sites in EOB during last scan but no longer emitting	PROCES
ONTIME	Compute intercept points to get back on path on time	AVOID
PIPNOW	Initialize RWW to pre-IP point for attack against site	ATTPTS
PREATT	Designates highest priority site for attack and weapon.	GATHER, SUPPRT
PRICOMP	Calculate attack and defensive priority values for site	CPRIOR
PROCES	Combines emitters into sites, generate new site EOB	APR38
PUCKER	Determine if attack should be aborted by crew	ATSTAT, SUPPRT
PULUP	Initialize RWW to pull-up point for AGM45 attack	ATTPTS
RANGE	Sets range quality used in calculation of defense & attack priorities	PROCES
REACT	Compute stop turn & end position for avoiding high threat	AVOID, GETHOME
RECOVR	Compute recovery point for recovery after attack on site	AGM65, PULUP
REJOIN	Stub (for future implementation of separate wing a/c)	BEAR
RELNOW	Initializes RWW to release point for attack on site	ATTPTS
RNGSET	Sets range value of site dep on a/c in lethal range or rot	A65PRI, CPCOMP, DEFSECT, PRICOMP
ROLIN	Initialize RWW to roll-in point for attack on site	ATTPTS
ROLLOFF	Initialize RWW to roll-off point for attack on site	ATTPTS
ROLLOUT	Initialize RWW to roll-out point for attack on site	ATTPTS
RTB	Sets return to base flag, computes return to base heading & next break point	STATUS
SAM	Sets mode of operation for SAM processed into site EOB	PROCES
SCOUNT	Updates dotted parameter, # of scans, site on or off air	PROCES
SITDEL	Undo kills made by already destroyed site	WWDRIV

LIST OF REACTIVE WILD WEASEL MODEL (RWWM) MODULES		
Name	**Function**	**Called by**
SITES	Changes to site tracking	LETHEX
ST5PRT	Default action for incomplete update of scan table	APR38
STATUS	Sets status flags for fuel, weapons, time on target	BEAR
STNAME	Returns name of scan table for given RWW & table #	
STPROS	Updates scan table of APR-38 EOB for intermed update	APR38
SUPPRT	Makes decisions for RWW in support role	BEAR
SYSSET	Assign value to system type parameter for attack priority	A65PRI, DEFSECT, PRICOMP
TANGPT	Compute points of tangency for different types of turns	AGM45, AGM65, GETBACK, REACT, RECOVR, TYP15
THISPT	Updates RWW & wingman to next point & updates path	LETHEX
THRTSF	Determine site threat and set threat value	
TIMING	Determine start time for scan table and cycle updates	APR38
TOCORS	Initializes RWW for attempt to get back to course	WWDRIV
TRACK	Checks dB for site missile & sets TRACK parameter	PROCES, SAM
TRNTYP	Compute turn type to get desired location & heading	AGM65, GETBACK
TYPE15	Processes for event type 15, calls other subroutines	LETHEX
UPDATE	Update Az, El, & Range for each site in EOB	OLDSIT, EOBGEN
WEAPON	Selects weapon for aircraft to use against site	PRICOMP
WEPDEC	Decrements weapons carried when weapon released	RELNOW
WEPNUM	Lookup table to prioritize weapons using tgt az & range	PRICOMP
WESDEL	Deletes all events for killed RWW	FUELUP, WWDRIV
WINCHR	Checks for Winchester condition for a/c	WEPDEC
WWDRIV	RWW driver module, event list processor	LETHEX

Table B-5, RWWM Outputs

RWWM OUTPUTS		
Variable Name	**Data Representation**	**Description**
AFPXX.OUT	Set of files "AFPxx.OUT" where XX is the Wild Weasel number	Actual flight path of each Wild Weasel or flight, and description of each break point
APRXX.OUT	Set of files "APRxx.OUT" where XX is the Wild Weasel number	Emitter information encountered by each Wild Weasel or flight

Table B-6, Candidate Measures of Effectiveness (MOEs)

CANDIDATE MEASURES OF EFFECTIVENESS (MOEs)	
MOE	**Calculated by**
Number of ground sites destroyed	Number of Event Type 2 occurrences
Number of aircraft destroyed	Number of Event Type 1 occurrences

Table B-7, RWWM State Variables

RWWM STATE VARIABLES		
Variable Name	**Description**	**Data Representation**
ACCODE	Current mode of operation	Array indexed by Wild Weasel number
AIRSPD	Current airspeed	Array indexed by Wild Weasel number
CLMRATE	Current climb rate	Array indexed by Wild Weasel number
FUEL1	Fuel flags	Array indexed by Wild Weasel number
BPSAVD	Break points saved	Array indexed by Wild Weasel number
ABPNUM	Actual break point number	Array indexed by Wild Weasel number
LBPNUM	Last break point number	Array indexed by Wild Weasel number
LINES	Line count before new heading required	Array indexed by Wild Weasel number
COLHED	Column heading flags	Array indexed by Wild Weasel number
HEDBCK	Heading back flag	Array indexed by Wild Weasel number
DLVDWEP	Delivered weapons flags	Array indexed by Wild Weasel number, site number and 1..2
ALIVE	Flags to indicate site status	Array indexed by site number
EFACTR	Wild Weasel effectiveness factor	Array indexed by Wild Weasel number
WESPS	Wild Weasel Ps	Array indexed by Wild Weasel number

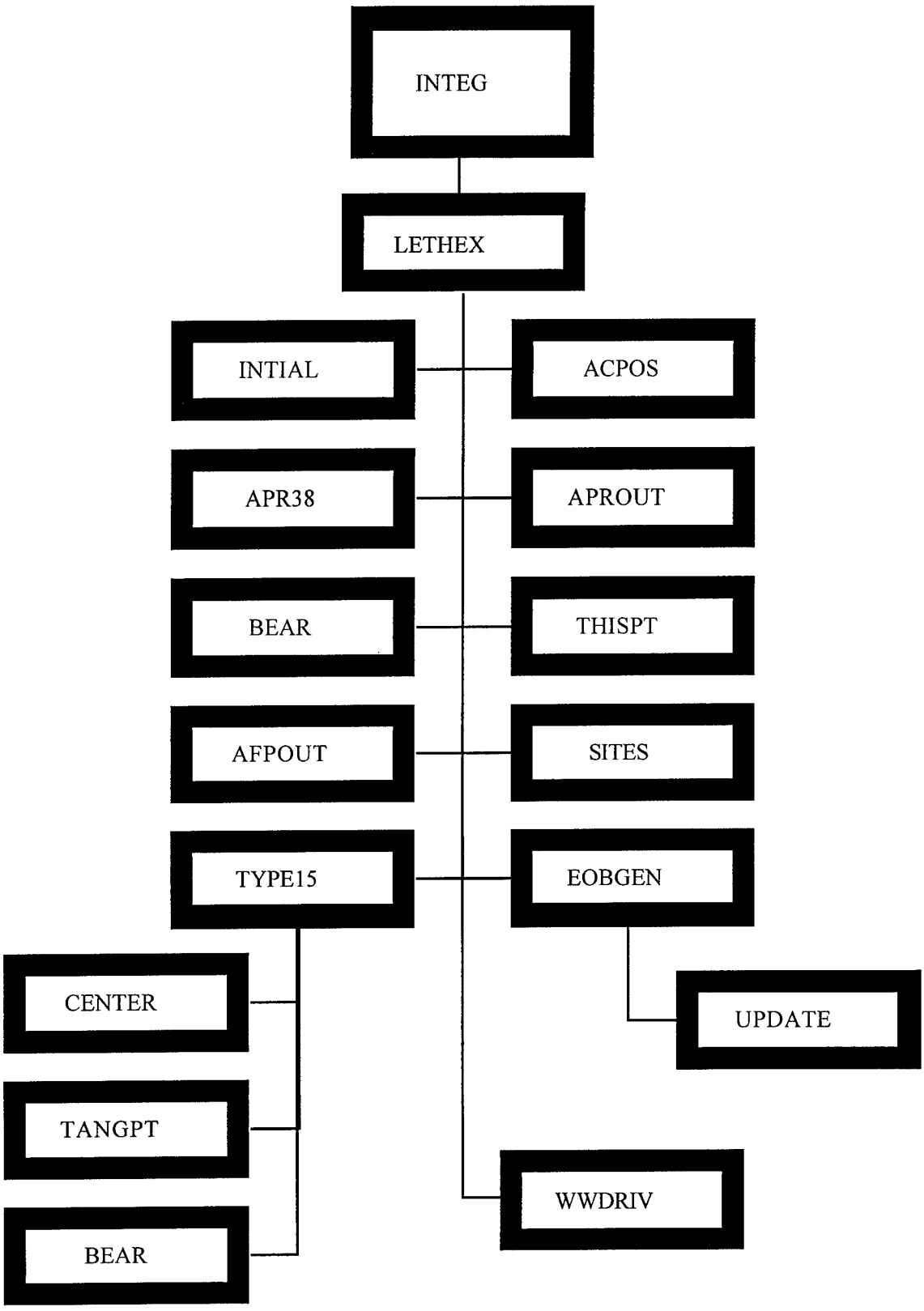


Figure B-1, RWWM Main (INTEG/LETHEX) Calling Structure

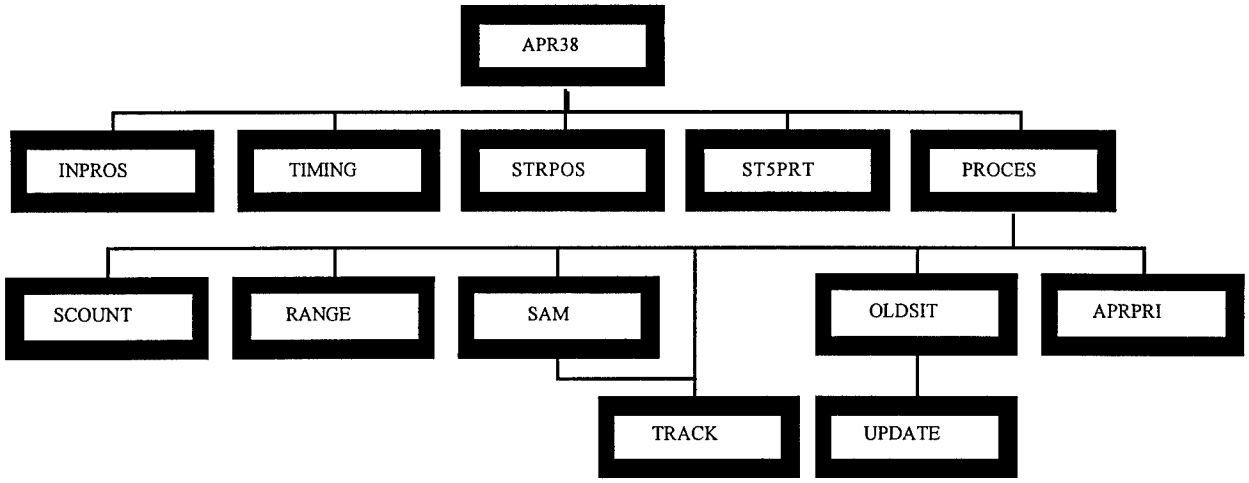


Figure B-2, APR38 Calling Structure

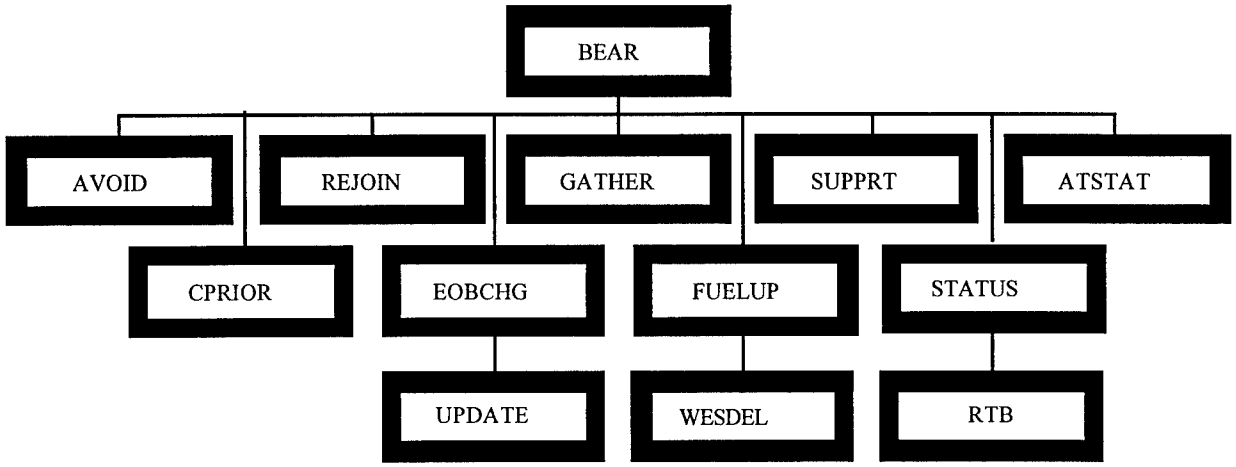


Figure B-3, BEAR Calling Structure

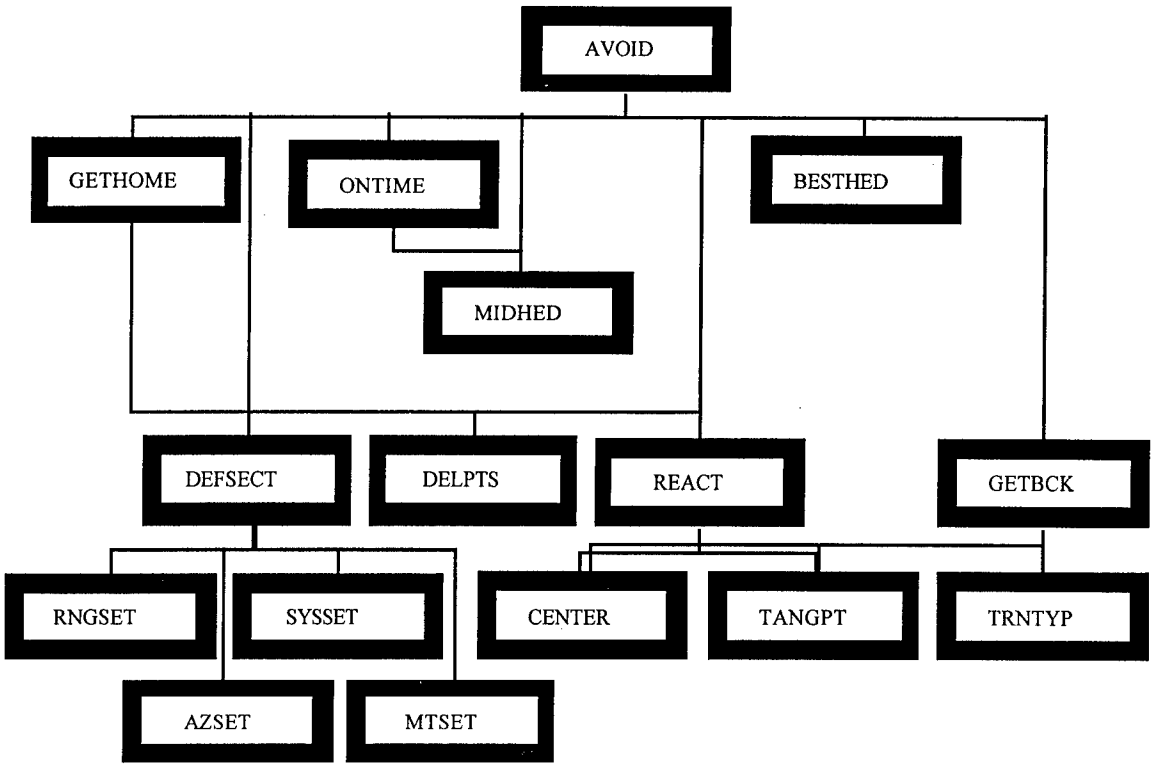


Figure B-4, AVOID Calling Structure

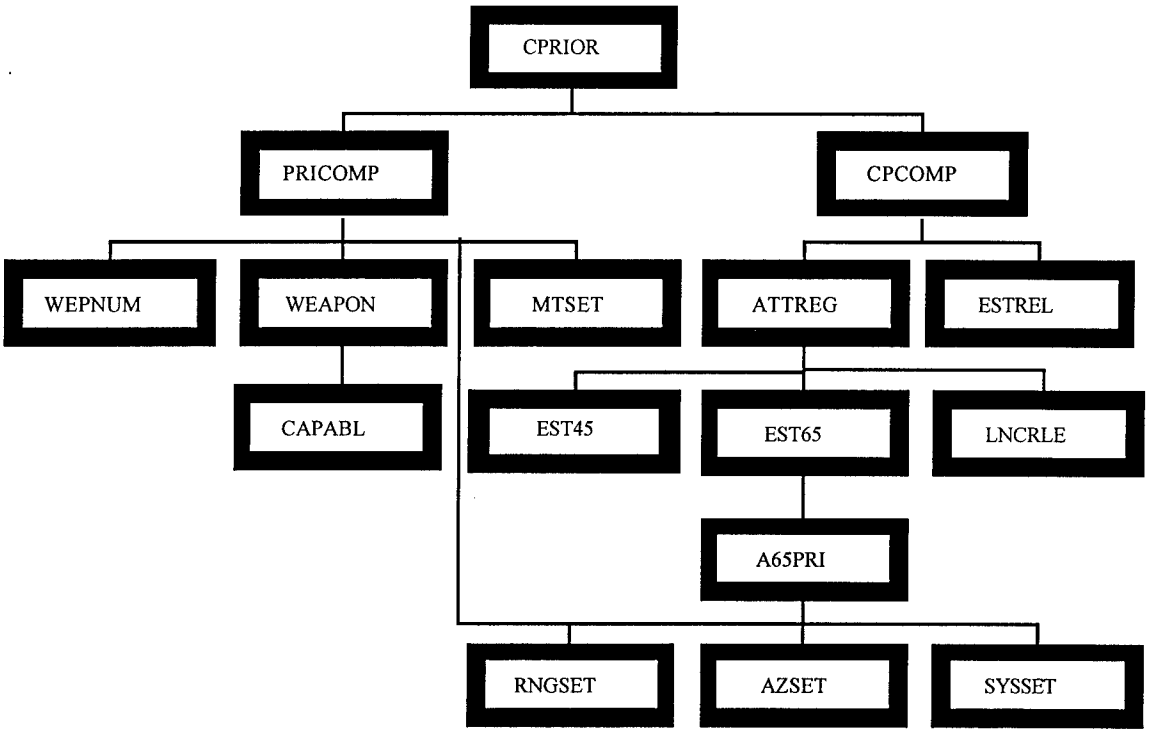


Figure B-5, CPRIOR Calling Structure

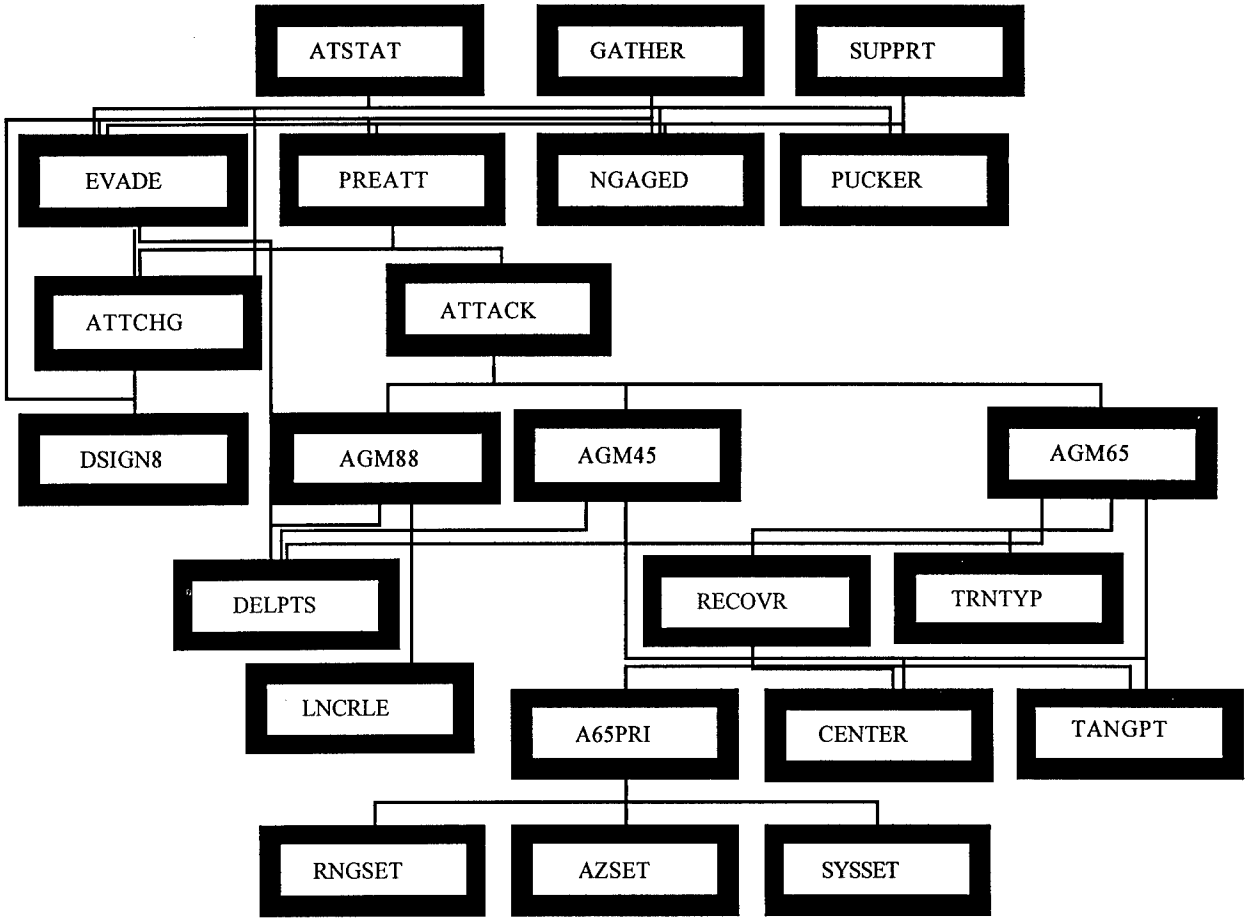


Figure B-6, GATHER/ATSTAT/SUPPRT Calling Structure

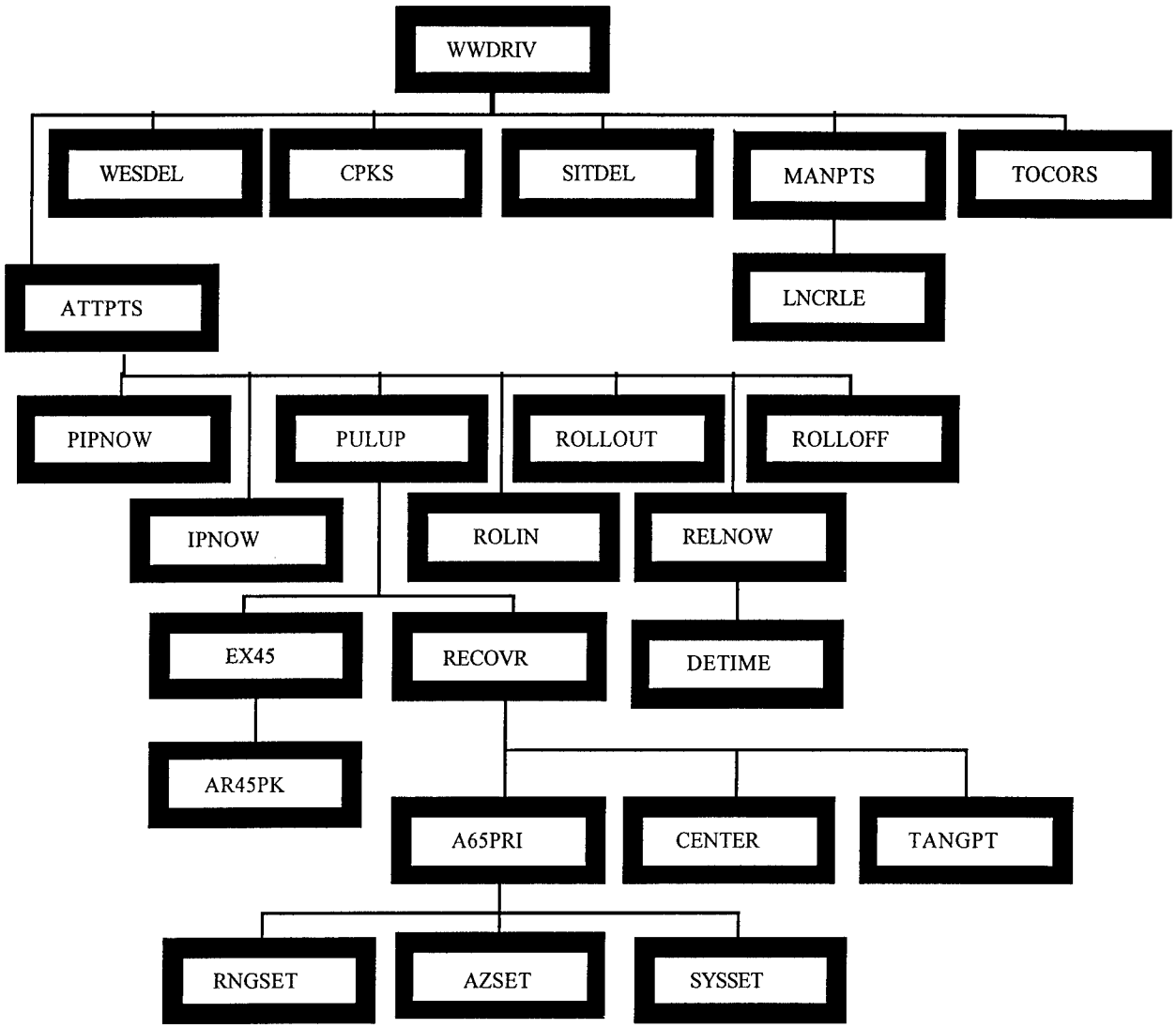


Figure B-7, WWDRIV Calling Structure

C. Detailed Derivation of RWWM Launch Point Aggregation

The following is the complete derivation of the formulas for launch computation abstraction described in Section 6.2.2.1.1.2. Rather than include the complete step by step derivation in the main body of the report, only the major steps are shown. The specific detailed sequence of steps is listed here.

We need to find T such that:

$$|w-r|<1000$$

We consider the problem as a two-dimensional problem, in which the dimensions are altitude and range. Altitude and range can be expressed in terms of the sine and cosine of the launch angle, which is constantly increasing (or decreasing) at the rate defined by p. Thus

$$l = pt$$

$$a = a_0 + \int_0^T s * \sin(pT)dt$$

$$a = a_0 + \left(s * \frac{-1}{p}\cos(pt)\right]_0^T \right)$$

$$a = a_0 + \left(s * \left(\left(\frac{-1}{p}\cos(pT) \right) - \left(\frac{-1}{p}\cos(0) \right) \right) \right)$$

$$a = a_0 + \left(s * \left(\frac{-\cos(pT)}{p} + \frac{1}{p} \right) \right)$$

$$a = a_0 + \left(\frac{s}{p} * (1-\cos(pT)) \right)$$

Substituting into the original equation for w

$$w = 1.789 * a + 986 * \deg(l) + 25766.22$$

(where deg(x) indicates a function to convert radians to degrees) we have

$$w = 1.789a_0 + \frac{1.789s}{p}(1-\cos(pT)) + 986\left(\frac{180}{\pi}\right)pT + 25766.22$$

Range is expressed as

$$r = r_0 - \int_0^T s\cos(pT)dt$$

which reduces to

$$r = r_0 - \frac{s}{p}\sin(pT)$$

Substituting back into the original equation, we must solve for T such that:

$$\left|1.789a_0+\frac{1.789s}{p}(1-\cos(pT))+986\left(\frac{180}{\pi}\right)pT+25766.22-\left(r_0-\frac{s}{p}\sin(pT)\right)\right|=1000$$

All parameters except T can be treated as constants in this calculation, so we move all of the constant terms to one side of the equation, leaving:

$$-\alpha\cos(pT)+\beta\sin(pT)+\gamma pT=\kappa$$

where

$$\alpha=\frac{1.789s}{p}$$

$$\beta=\frac{s}{p}$$

$$\gamma=986\left(\frac{180}{\pi}\right)$$

$$\kappa=r_0-1.789a_0-\frac{1.789s}{p}-25766.22\pm1000$$

Over the range 0° to 45°, the sine and cosine terms of the above equation can be approximated by a straight line from

$$(0,-\alpha),\left(\frac{\pi}{4},\beta\sin(45^\circ)-\alpha\cos(45^\circ)\right)$$

The y-intercept of this line is -α, while the slope is

$$=\frac{\Delta y}{\Delta x}$$

$$=\frac{(\beta\sin(45^\circ)-\alpha\cos(45^\circ))-(-\alpha)}{\frac{\pi}{4}-0}$$

$$=\frac{\left(\beta\left(\frac{\sqrt{2}}{2}\right)-\alpha\left(\frac{\sqrt{2}}{2}\right)\right)+\alpha}{\frac{\pi}{4}}$$

$$=\frac{\frac{\sqrt{2}}{2}(\beta-\alpha)+\alpha}{\frac{\pi}{4}}$$

More specifically, we substitute into the slope-intercept expression of a straight line

$$y = mx + b$$

to generate

$$\frac{\frac{\sqrt{2}}{2}(\beta - \alpha) + \alpha}{\frac{\pi}{4}} pT - \alpha$$

for the sine and cosine terms. Substituting this expression for the cosine and sine terms, we now have an approximate expression for T.

$$\frac{\frac{\sqrt{2}}{2}(\beta-\alpha)+\alpha}{\frac{\pi}{4}}pT-\alpha+\gamma pT\approx\kappa$$

$$\frac{\frac{\sqrt{2}}{2}(\beta-\alpha)+\alpha}{\frac{\pi}{4}}pT+\gamma pT\approx\kappa+\alpha$$

$$pT\left(\frac{\frac{\sqrt{2}}{2}(\beta-\alpha)+\alpha}{\frac{\pi}{4}}+\gamma\right)\approx\kappa+\alpha$$

$$pT\approx\frac{\kappa+\alpha}{\frac{\frac{\sqrt{2}}{2}(\beta-\alpha)+\alpha}{\frac{\pi}{4}}+\gamma}$$

$$T\approx\frac{\kappa+\alpha}{p\left(\gamma+\frac{\frac{\sqrt{2}}{2}(\beta-\alpha)+\alpha}{\frac{\pi}{4}}\right)}$$

$$T\approx\frac{\kappa+\alpha}{p\left(\gamma+\frac{\frac{\sqrt{2}}{2}(\frac{s}{p}-1.789\frac{s}{p})+1.789\frac{s}{p}}{\frac{\pi}{4}}\right)}$$

$$T \approx \frac{\kappa + \alpha}{p\left(\gamma + \dfrac{\left(\dfrac{s}{p}\right)\left(\dfrac{\sqrt{2}}{2}(-.789) + 1.789\right)}{\dfrac{\pi}{4}}\right)}$$

$$T \approx \frac{\kappa + \alpha}{p\gamma + p\left(\dfrac{\left(\dfrac{s}{p}\right)\left(1.789 - \dfrac{\sqrt{2}}{2}(.789)\right)}{\dfrac{\pi}{4}}\right)}$$

Expressed in terms of the original values, T is approximated as

$$T \approx \frac{r_0 - 1.789a_0 - 2576.22 \pm 1000}{\dfrac{986 * 180}{\pi}p + \dfrac{1.789 - \dfrac{\sqrt{2}}{2} * (.789)}{\dfrac{\pi}{4}}s}$$

Now consider the other approximation for the sine and cosine terms in the equation

$$-\alpha\cos(pT) + \beta\sin(pT) + \gamma pT = \kappa$$

Using the identity

$$M\sin(Bt) - N\cos(Bt) = A\sin(Bt - C)$$

where

$$A = \sqrt{M^2 + N^2} \text{ and } \sin C = \frac{N}{\sqrt{M^2 + N^2}}$$

we replace the sine and cosine terms with the term

$$\sqrt{\alpha^2 + \beta^2}\sin(pT - C) \text{ where } C = \sin^{-1}\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}$$

We can use the same straight line approximation as used above for this sine curve over the interval $\left(0, \frac{\pi}{4}\right)$. To show that the straight line approximation is identical to the one used above, we show that the end points are identical. The following calculations show the values of the sine term at $pT = 0$ and $pT = \frac{\pi}{4}$. For $pT = 0$

$$y = \sqrt{\alpha^2 + \beta^2}\,\sin\left(0 - \sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)$$

$$y = \sqrt{\alpha^2 + \beta^2}\,\sin\left(-\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)$$

$$y = -\sqrt{\alpha^2 + \beta^2}\,\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)$$

$$y = -\sqrt{\alpha^2 + \beta^2}\,\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}$$

$$y = -\alpha$$

For $pT = \frac{\pi}{4}$

$$y = \sqrt{\alpha^2 + \beta^2}\,\sin\left(\frac{\pi}{4} - \sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)$$

Using the identity $\sin(x - y) = \sin x \cos y - \sin y \cos x$,

$$y = \sqrt{\alpha^2 + \beta^2}\left(\sin\frac{\pi}{4}\cos\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right) - \cos\frac{\pi}{4}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)\right)$$

First evaluate the $\sin\left(\frac{\pi}{4}\right)$ and $\cos\left(\frac{\pi}{4}\right)$

$$y = \sqrt{\alpha^2 + \beta^2}\left(\frac{\sqrt{2}}{2}\cos\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right) - \frac{\sqrt{2}}{2}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)\right)$$

Then using the identity $\cos\left(\sin^{-1} x\right) = \sqrt{1 - x^2}$

$$y = \sqrt{\alpha^2 + \beta^2}\left(\frac{\sqrt{2}}{2}\sqrt{1 - \left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)^2} - \frac{\sqrt{2}}{2}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)\right)$$

$$y = \sqrt{\alpha^2 + \beta^2}\left(\frac{\sqrt{2}}{2}\sqrt{1 - \frac{\alpha^2}{\alpha^2 + \beta^2}} - \frac{\sqrt{2}}{2}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}\right)\right)\right)$$

$$y = \sqrt{\alpha^2+\beta^2}\left(\frac{\sqrt{2}}{2}\sqrt{\frac{\alpha^2+\beta^2}{\alpha^2+\beta^2}-\frac{\alpha^2}{\alpha^2+\beta^2}}-\frac{\sqrt{2}}{2}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2+\beta^2}}\right)\right)\right)$$

$$y = \sqrt{\alpha^2+\beta^2}\left(\frac{\sqrt{2}}{2}\sqrt{\frac{\alpha^2+\beta^2-\alpha^2}{\alpha^2+\beta^2}}-\frac{\sqrt{2}}{2}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2+\beta^2}}\right)\right)\right)$$

$$y = \sqrt{\alpha^2+\beta^2}\left(\frac{\sqrt{2}}{2}\sqrt{\frac{\beta^2}{\alpha^2+\beta^2}}-\frac{\sqrt{2}}{2}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2+\beta^2}}\right)\right)\right)$$

$$y = \sqrt{\alpha^2+\beta^2}\left(\frac{\sqrt{2}}{2}\frac{\beta}{\sqrt{\alpha^2+\beta^2}}-\frac{\sqrt{2}}{2}\sin\left(\sin^{-1}\left(\frac{\alpha}{\sqrt{\alpha^2+\beta^2}}\right)\right)\right)$$

Now reducing the last term of the equation

$$y = \sqrt{\alpha^2+\beta^2}\left(\frac{\sqrt{2}}{2}\frac{\beta}{\sqrt{\alpha^2+\beta^2}}-\frac{\sqrt{2}}{2}\frac{\alpha}{\sqrt{\alpha^2+\beta^2}}\right)$$

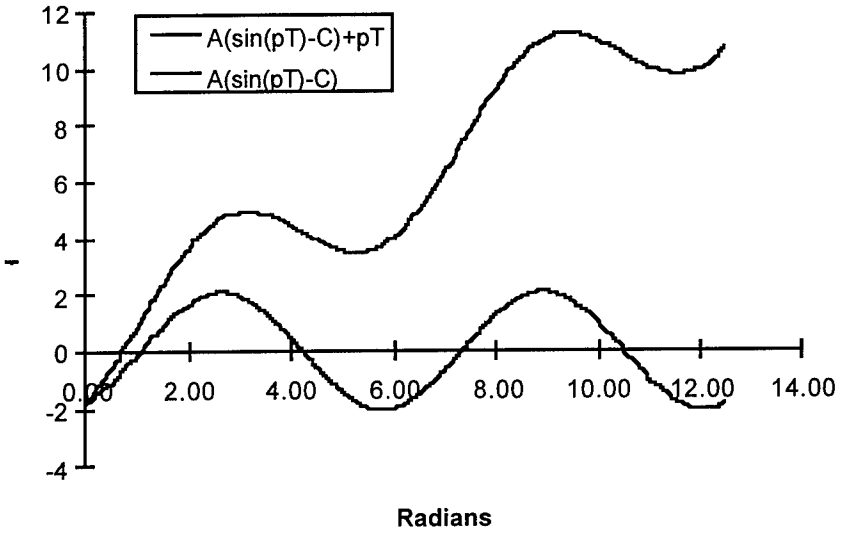
$$y = \sqrt{\alpha^2+\beta^2}\left(\frac{\sqrt{2}}{2}\frac{\beta-\alpha}{\sqrt{\alpha^2+\beta^2}}\right)$$

$$y = \frac{\sqrt{2}}{2}\beta-\alpha$$

Thus we see that the substitution of the $A(\sin Bt - C)$ form of the equation yields the same end points for the straight line approximation. The calculation of the slope-intercept form of the approximation and the substitution back into the original equation for T are identical to that described above.

Unfortunately, applying this identity to eliminate the cosine term does not appear to help us. A small angle approximation $\sin\theta \approx \theta$ is inappropriate because of the phase shift term (-*C*). That term is equal to the arctangent of 1.789, which is approximately 60°. Thus The actual range of values of the sine term for $pT \in [0°,45°]$ is approximately $[-60°,-15°]$.

The mathematical relationships are plotted below.


A(sin(pT)-C)+pT
A(sin(pT)-C)
12
10
8
6
4
2
0
-2
-4
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
Radians

D. ALARM Parameter Dependency Graphs

As part of the analysis of the ALARM model, parameter dependency graphs (PDGs) were developed to depict the causal ordering of parameters in the model. These graphs can be used to partition the model and identify submodel components, for which abstractions could be investigated.

In Section 7.2.2 of the report, the top-level PDG was included to illustrate the concept of the PDG. In this appendix, the full set of PDGs is presented, along with a table listing information about the parameters. Because there are a number of cases in which input parameters are used in multiple processing threads, we did not attempt to depict them graphically. Instead, the information is included in the accompanying table, which shows the origin of the parameters, and where the parameters are used.

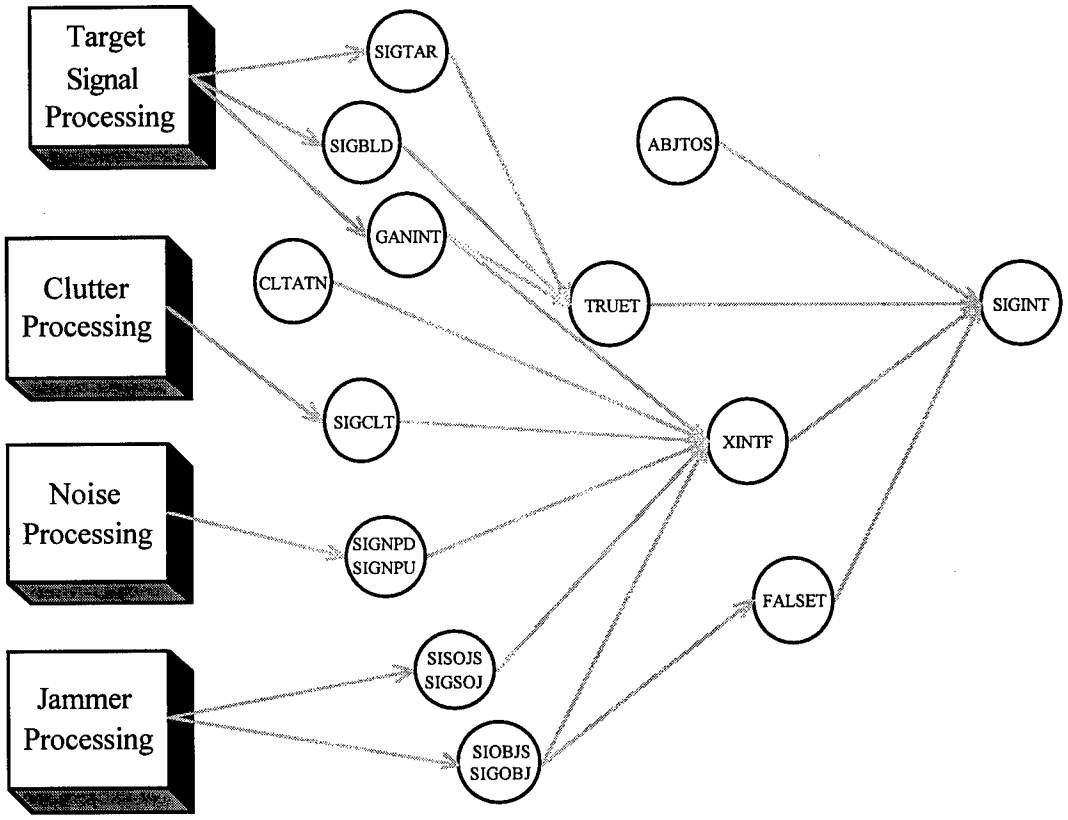


Figure D-1, ALARM Top Level Parameter Dependency Graph

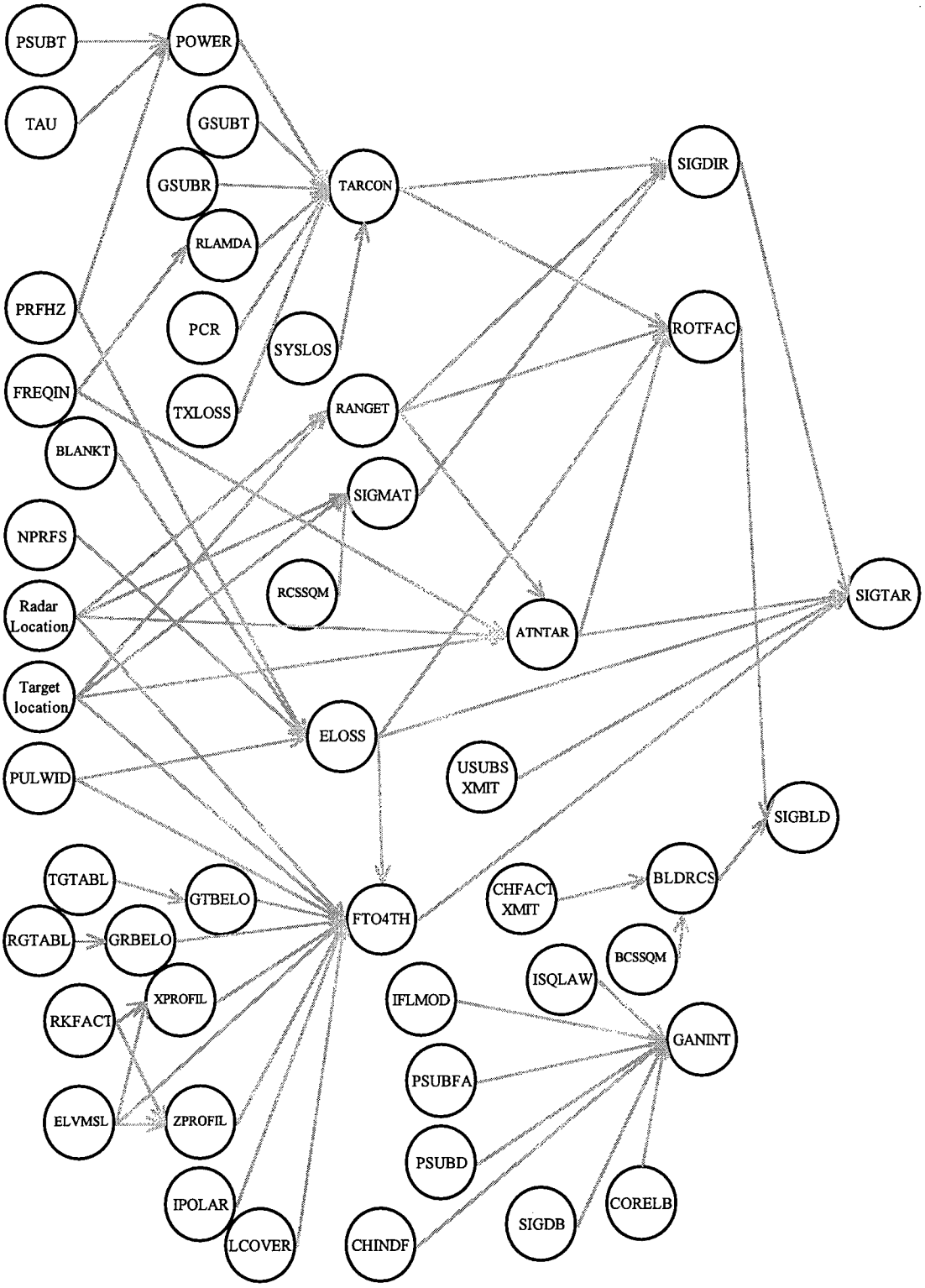


Figure D-2, ALARM Target Processing Parameter Dependency Graph

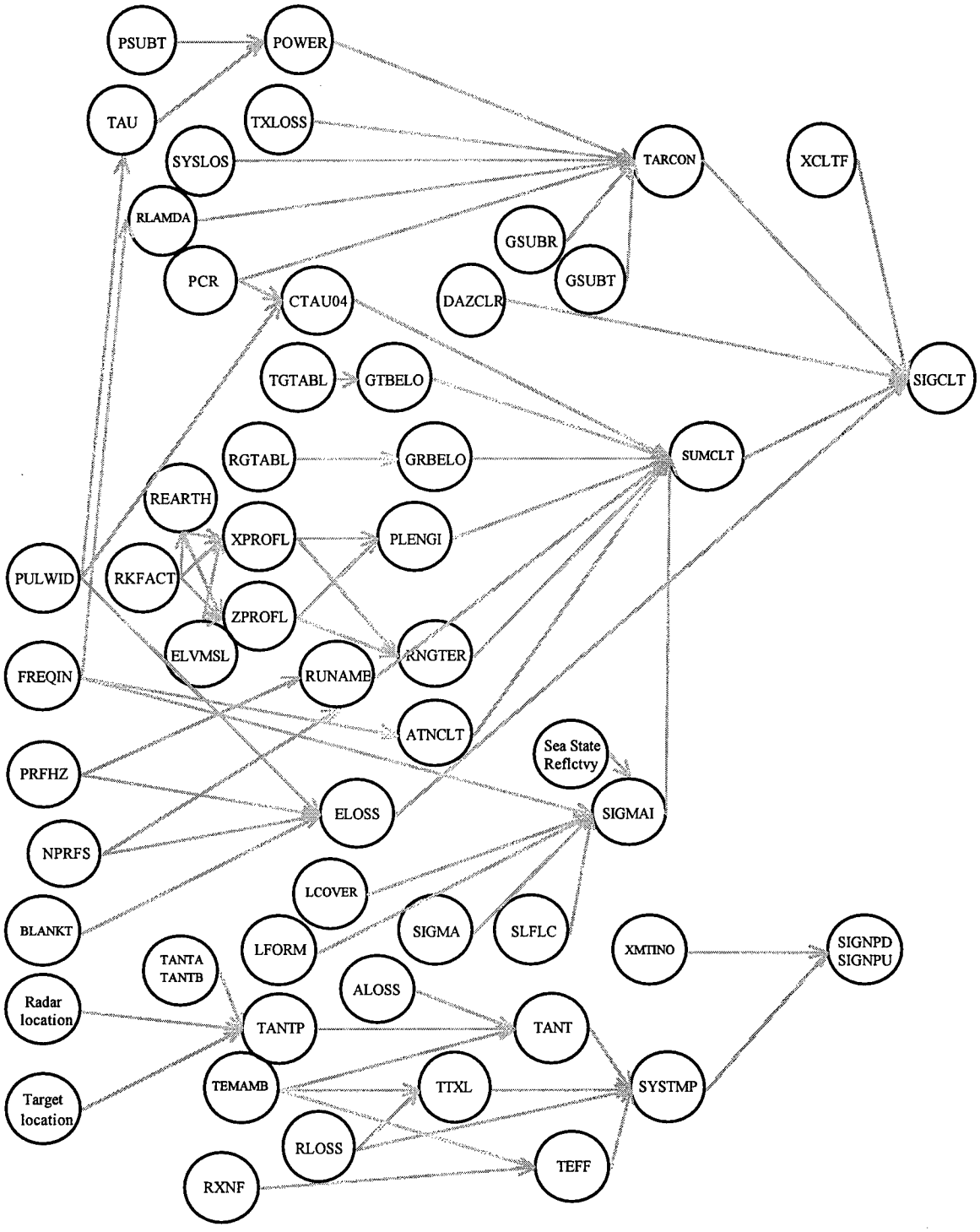


Figure D-3, ALARM Clutter and Noise Processing Parameter Dependency Graph

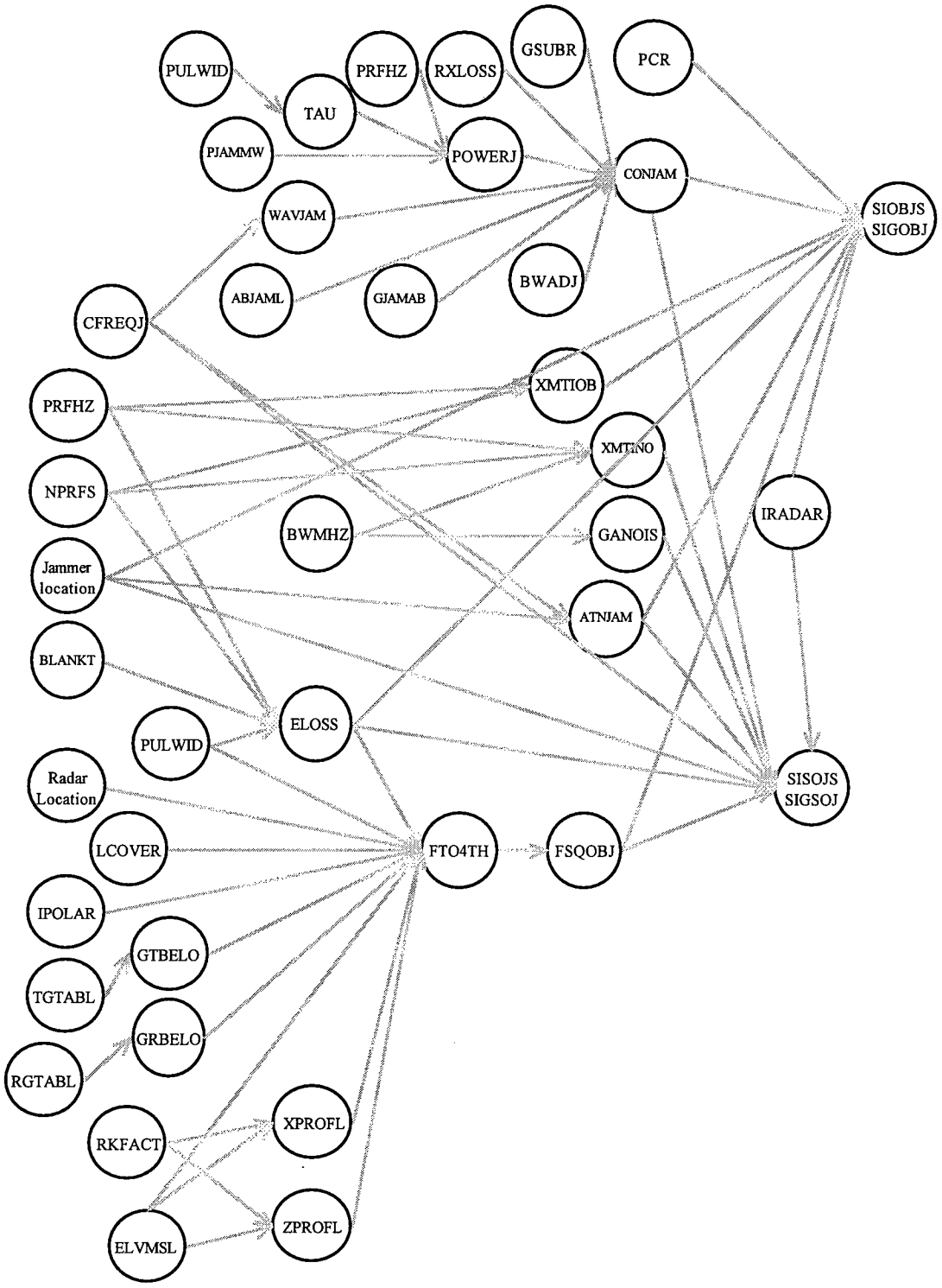


Figure D-4, ALARM Jamming Processing Parameter Dependency Graph

Table D-1, ALARM Parameters

ALARM Parameters				
Parameters	Symbol	Description	Proc. Thread	Source
ABJAML	L_{OBJ} L_{SOJj}	Jammer transmit loss (for onboard jammer or standoff jammers)	Jamming	DATAJAMR
ABJTOS	J/S_t	Jamming-to-signal ratio necessary for the deception jamming to be effective	Top-Level	DATAJAMR
ALOSS	L_{ant}	Antenna ohmic loss	Noise	DATARADR
ATNCLT	A_{ci}	Atmospheric attenuation between the radar and the clutter patch	Clutter	Computed
ATNJAM	A_{OBJ} A_{SOJj}	Atmospheric attenuation from the radar to an onboard jammer	Jamming	Computed
ATNTAR	A_a	Atmospheric attenuation from the radar to the target	Target	Computed
BCSSQM	σ_j	Radar cross section of each rotor blade doppler shift/radar cross section pair	Target	DATAROTO
BLANKT	T_b	Transmitter blanking time after pulse transmission	Target Clutter Jamming	DATARADR
BLDRCS	σ_{Bi}	Sum of the signal processing reponse times the rotor blade radar cross section	Target	Computed
BWADJ	B_{adjj} B_{adjj}	Ratio of the radar receiver bandwidth to the jammer bandwidth as a function of spectral overlap (for onboard jammer or standoff jammers)	Jamming	Computed from data in DATAJAMR
BWMHZ	B_{IF}	Receiver intermediate frequency bandwidth	Jamming	DATARADR
CFREQJ	f_{OBJ} f_{SOJj}	Onboard jammer transmit frequency	Jamming	DATAJAMR
CHFACT	G_{spij}	Response of the frequency domain signal processing to the doppler frequency associated with each target rotor blade for pulse doppler radar	Target	Computed
CHINDF		Number of degrees of freedom for a Chi squared or Weinstock model	Target	DATARCST
CLTATN	G_{sp}	Response of the frequency domain signal processing to the clutter signal PSD for a pulsed radar	Top-Level	Computed
CONJAM	C_{Ji} C_{Jij}	Constant comprised of all constant terms used to determine jammer signal power (separate computations for onboard or standoff jammers)	Jamming	Computed
CORELB		Number of correlated blocks for a Chi squared model	Target	DATARCST
CTAUO4	ΔR	Radar range resolution	Clutter	Computed
DAZCLR	$\Delta\Theta_c$	Angular width of the clutter patch	Clutter	DATARADR
ELOSS	L_{ei}	Pulse blanking and eclipsing loss	Target Clutter	Computed
ELVMSL		Terrain data	Target Clutter Jamming	Terrain data base
FALSET	S_{FTi}	False target signal power	Top-Level	Computed
FREQIN	f_0	Radar transmit frequency	Target	DATARADR

ALARM Parameters				
Parameters	Symbol	Description	Proc. Thread	Source
FSQOBJ	F_{OBJ}^2 F_{SOJj}^2	Propagation factor (calculated for onboard jammer or standoff jammer)	Jamming	Computed
FTO4TH	F^4	Target pattern propagation factor	Target Jamming	Computed
GANINT	G_I	Integration gain	Top-Level	Computed
GANOIS	B_{ni}	Receiver noise bandwidth for a pulse doppler radar	Jamming	Computed
GJAMAB	G_{OBJ} G_{SOJj}	Jammer transmit antenna mainbeam gain (for onboard jammer or standoff jammers)	Jamming	DATAJAMR
GRBELO	G_{rc0}	Radar receive antenna gain relative to the mainbeam gain in the direction of the clutter patch	Target Clutter Jamming	Computed
GSUBR	G_r	Radar receieve antenna mainbeam gain	Target Clutter Jamming	DATAGANR
GSUBT	G_t	Radar transmit antenna mainbeam gain	Target Clutter	DATAGANT
GTBELO	G_{tc0}	Radar transmit antenna gain relative to the mainbeam gain in the direction of the clutter patch	Target Clutter Jamming	Computed
IFLMOD		Target flucuation model	Target	DATARADR
IPOLAR		Antenna polarization	Target Jamming	DATARADR
IRADAR		Type of radar (pulsed or pulse doppler)	Jamming	DATARADR
ISQLAW		Radar envelope detector type	Target	DATARADR
Jammer location		Latitude/longitude coordinates of the jammers	Jamming	DATAJAMR
LCOVER		Land cover index used to retrieve clutter coefficient	Target Clutter Jamming	DATAREFL
LFORM		Land form index used to retrieve clutter coefficient	Clutter	DATAREFL
NPRFS		Number of pulse repetition frequencies	Target Clutter Jamming	DATARADR
PCR PCRFAC	 G_p PCR_f	Pulse compression ratio	Target Clutter Jamming	DATARADR
PJAMMW	P_{OBJ} P_{SOJj}	Jammer peak transmitter power (for onboard jammer or standoff jammers)	Jamming	DATAJAMR
PLENGI	τ_{cI}	Illuminated length of the clutter patch on the ground based on the radar pulse length and terrain masking	Clutter	Computed
POWER	P_i	Radar transmitter power	Target Clutter	Computed
POWERJ	P_i P_{ij}	Jammer transmitter power (for onboard jammer or standoff jammers)	Jamming	Computed

ALARM Parameters

Parameters	Symbol	Description	Proc. Thread	Source
PRFHZ		Radar pulse repetition frequencies	Target Clutter Jamming	DATARADR
PSUBD		Probability of detection	Target	DATARADR
PSUBFA		Probability of false alarm	Target	DATARADR
PSUBT	P_t	Radar peak transmitter power	Target Clutter	DATARADR
PULWID		Radar pulse width	Target Clutter Jamming	DATARADR
Radar location		Site location of the ground-based radar	Target Jamming	DATASITE
RANGET	R	Slant range from radar to target	Target	Computed
RCSSQM		Target radar cross section table	Target	DATARCST
REARTH		Earth radius	Target Clutter Jamming	Computed
RGTABL		Radar receive antenna pattern	Target Clutter Jamming	DATAGANR
RJAMSQ	R R_{SOJi}	Slant range from the radar to the onboard jammer or standoff jammers	Jamming	Computed
RKFACT		Refractivity factor	Target Clutter Jamming	DATASEKE
RLAMDA	λ	Radar transmit wavelength	Target Clutter	Computed
RLOSS	L_r	Receieve line loss	Noise	DATARADR
RNGTER	R_{ci}	Slant range between the radar and the clutter patch	Clutter	Computed
ROTFAC	ROT_{fi}	Factor used in determining target rotor blade signal	Target	Computed
RUNAMB	R_{max}	Maximum unambiguous range	Clutter	Computed
RXLOSS	L_{rs}	Radar receieve and system losses	Jamming	DATARADR
RXNF	NF	Receiever noise figure	Noise	DATARADR
Sea State Reflc.		Sea clutter reflectivity percentiles	Clutter	Implemented in CNASIG code
SIGBLD	S_{Bi}	Rotating blade signal power	Top-Level Target	Computed
SIGCLT	S_{Ci}	Clutter signal power	Clutter	Computed
SIGDB		Sigma parameter of a log-normal distribution	Target	DATARCST
SIGDIR	S_{Tdi}	Target bidy signal power along direct path from radar to target	Target	Computed
SIGINT	S/I_i	Signal-to-interference ratio	Top-Level	Computed
SIGMA		Sea clutter coefficient table	Clutter	DATA statement in CNASIG
SIGMAI	σ_0	Reflectivity of the clutter patch	Clutter	Computed

ALARM Parameters				
Parameters	Symbol	Description	Proc. Thread	Source
SIGMAT	σ	Target body radar cross section	Target	Computed
SIGNPD	S_{Ni}	System noise signal power for pulse doppler radar	Top-Level Noise	Computed
SIGNPU	S_{Ni}	System noise signal power for pulsed radar	Top-Level Noise	Computed
SIGOBJ	S_{JOi}	Onboard jammer signal power for a pulsed radar	Top-Level Jamming	Computed
SIGSOJ	S_{JSi}	Standoff jammer signal power for a pulsed radar	Top-Level Jamming	Computed
SIGTAR	S_{Ti}	Target body signal power	Top-Level Target	Computed
SIOBJS	S_{JOi}	Onboard jammer signal power for a pulse doppler radar	Top-Level Jamming	Computed
SISOJS	S_{JSi}	Standoff jammer signal power for a pulsed doppler radar	Top-Level Jamming	Computed
SLFLC		Land clutter coefficient table	Clutter	DATA statement in MTISIG
SUMCLT	C_{sum}	Sum of the varying components which comprise the clutter signal from all clutter patches	Top-Level Clutter	Computed
SYSLOS	L_s	Radar system loss	Target Clutter	DATARADR
SYSTMP	T_{sys}	System noise temperaure	Noise	Computed
TANT	T_{ant}	Antenna noise temperature	Noise	Computed
TANTA TANTB		Antenna noise temperature table	Noise	DATA statement in SYSNOI
TANTP	T_{antp}	Lossless antenna noise temperature	Noise	Computed
TARCON	C_{Ti}	Constant comprised of all the constant terms used to determine the target bidy signal level	Target Clutter	Computed
Target location		Target way points	Target	DATATARG (for flight path mode)
TAU	τ	Radar transmitted pulse length	Target Clutter Jamming	Computed
TEFF	T_{eff}	Receiever noise equivalent temperature	Noise	Computed
TEMAMB	T_{amb}	Ambient thermal temperature	Noise	PARAMETER statement in SYSNOI
TGTABL		Radar transmit antenna pattern.	Target Clutter Jamming	DATAGANT
TRUET	S_{TTi}	True target signal power	Top-Level	Computed
TTXL	T_{txl}	Receive transmission line noise temperature	Noise	Computed
TXLOSS	L_t	Radar transmit line loss	Target Clutter	DATARADR

ALARM Parameters				
Parameters	Symbol	Description	Proc. Thread	Source
USUBS	G_{sp}	Response of the frequency domain signal processing to the doppler frequency of the target body for pulse doppler radar	Target	Computed
WAVJAM	λ_{OBJ} λ_{SOJj}	Jammer wavelength (for onboard jammer or standoff jammers)	Jamming	Computed
XCLTF	G_{Csp} G_{spi}	Response of the doppler filer which contains the target signal to the clutter power spectral density	Clutter	Computed
XINTF	I_i	Interference signal power	Top-Level	Computed
XMIT	G_{sp} G_{spij}	Response of the frequency domain signal processing to the doppler frequency of the target body for pulsed radar	Target	Computed
XMTINO	G_{sp}	Response of the frequency domain signal processing over the receiver noise bandwidth for a pulsed radar.	Noise	Computed
XMTIOB	G_{spi}	Response of the frequency domain signal processing to the onboard jammer signal	Jamming	Computed
XPROFL		X-ccordinates of terrain profile between radar and target	Target Clutter Jamming	Computed
ZPROFL		Z-coordinates of terrain profile between radar and target	Target Clutter Jamming	Computed

E. TERSM Metamodel Graphs

This appendix includes graphs of TERSM metamodel outputs to help illustrate the metamodel results. The first step in using the metamodels was to write a program to generate metamodel outputs for various test cases, where each test case is a set of four parameter values (altitude, velocity, azimuth, and channel capacity). Ideally, we could confirm that the program generated the correct values if we had comparable outputs from the original study. However, that data was not available; instead, Zeimer et. al. (1993) included the Maximum Average Error (MAE) and the Average Absolute Relative Error (AARE) for each of the seven metamodels. (These metrics were used to demonstrate the relative fit of the metamodels.) To confirm our implementation of the published metamodel equations, we used a test data file containing the forty-nine (49) TERSM test cases used for the metamodel study, and generated MAE and AARE metrics for each of the seven metamodels.

The AARE metric generated using the published equations matched in only three of the seven metamodels, as shown in Table E-1. Metamodel 5 was examined first, because of its large variance and the fact that the MAE metric also did not match. Our MAE occurred for the test case (0,0,-1,0). Since only the x_3 input is nonzero, the metamodel reduces to

$$549.756 - 62.778x_3 - 22.962x_3^2$$

which for x_3 = -1 evaluates to 589.572. Since the TERSM output for this test case is 419, the absolute error of the metamodel is 170.572, which exceeds the value published in Table E-2 of Zeimer et. al. (1993). Upon further inspection, we realized that simply changing the sign of the x_3 term in the published metamodel equation yielded MAE and AARE values which matched the original values. We thus concluded that the metamodel development was correct, and that the error was in the documentation of the metamodel equation in the report. Following a similar analysis, we were able to generate matching MAE and AARE values by changing the sign of the $x_1x_2x_3$ term in Metamodel 4 and the x_1x_3 term in Metamodel 7.

We attempted to apply a similar analysis to the difference in error metrics generated in Metamodel 2. However, there were no cases in which simply changing the sign of the term yielded error metrics which matched the original values. We considered the possibility that perhaps two signs were in error, and wrote a program which generated all possible combinations of sign changes, using the same coefficients (2^{14} possible combinations). However, no error metrics were lower than the original metrics. At this point we concluded that there is an unknown error in the published metamodel equation.

The remainder of this paper uses the TERSM metamodels to illustrate some observations about metamodels. We use the corrected versions of Metamodels 4, 5, and 7. Because of our inability to completely replicate Metamodel 2, it is not used. Table E-2 lists the metamodels

which we used. Table E-3 lists the data points used to generate the metamodels, and the predicted results of the metamodels for each data point.[1]

Table E-1, Metamodel Error Metrics

Metamodel Error Metrics				
Metamodel	Published MAE	Calculated MAE from Published Eqs	Published AARE	Calculated AARE from Published Eqs
1	421.6	421.6	44.3%	44.3%
2	220.6	220.8	14.2%	15.0%
3	256.3	256.4	12.3%	12.3%
4	225.4	221.5	11.4%	13.9%
5	120.9	170.6	8.3%	41.3%
6	94.3	94.2	4.0%	4.0%
7	73.5	73.9	4.7%	8.9%

Table E-2, Metamodels

Metamodels	
Num.	Equation
1	$224.118 + 85.750x_2 + 57.750x_3 + 89.000x_4 + 27.750x_1x_4 + 21.000x_2x_3 + 47.250x_2x_4$
3	$6.346 - 0.047x_1 + 0.417x_2 + 0.306x_3 + 0.467x_4 - 0.025x_1x_2 + 0.075x_1x_3 + 0.170x_1x_4 + 0.098x_2x_4 - 0.028x_3x_4 + 0.066x_1x_2x_3 - 0.078x_2x_3x_4 - 0.049x_1x_2x_3x_4 - 0.133x_1^2 - 0.679x_2^2 - 0.125x_3^2 - 0.363x_4^2$
4	$23.319 + 2.994x_2 + 2.042x_3 + 3.288x_4 + 0.488x_1x_3 + 1.006x_1x_4 + 0.407x_2x_3 + 1.155x_2x_4 + 0.257x_3x_4 + 0.400x_1x_2x_3 - 0.288x_2x_3x_4 - 0.841x_1^2 - 5.757x_2^2 - 0.701x_3^2 - 2.683x_4^2$
5	$549.756 - 32.722x_1 - 111.537x_2 + 62.778x_3 + 180.556x_4 + 16.441x_1x_3 + 26.647x_1x_4 + 20.882x_2x_3 + 46.971x_2x_4 + 18.676x_3x_4 + 12.508x_1x_2x_3 + 9.785x_1x_2x_4 + 12.892x_1x_3x_4 + 61.972x_1^2 - 22.962x_3^2 - 86.491x_4^2 + 44.667x_1^3 + 203.481x_2^3 - 78.222x_4^3 - 90.454x_1^4 - 210.918x_2^4$
6	$6.350 - 0.047x_1 - 0.308x_2 + 0.075x_3 + 0.267x_4 - 0.022x_1x_2 + 0.073x_1x_3 + 0.162x_1x_4 + 0.021x_2x_3 + 0.099x_2x_4 - 0.029x_3x_4 + 0.066x_1x_2x_3 + 0.014x_1x_2x_4 - 0.078x_2x_3x_4 - 0.049x_1x_2x_3x_4 + 0.260x_1^2 - 0.125x_3^2 - 0.359x_4^2 + 0.725x_2^3 + 0.230x_3^3 + 0.199x_4^3 - 0.398x_1^4 - 0.682x_2^4$
7	$23.567 - 0.669x_1 - 2.842x_2 + 1.298x_3 + 3.344x_4 + 0.491x_1x_3 + 0.963x_1x_4 + 0.414x_2x_3 + 1.155x_2x_4 + 0.231x_3x_4 + 0.404x_1x_2x_3 + 0.198x_1x_2x_4 + 0.201x_1x_3x_4 - 0.285x_2x_3x_4 + 2.037x_1^2$

[1]The first metamodel used only the first seventeen (17) data points from TERSM, while models 2 through 4 used the first twenty-five (25) observations.

	$- 0.788x_3^2 - 2.743x_4^2 + 0.714x_1^3 + 5.836x_2^3 + 0.744x_3^3 - 2.947x_1^4 - 5.823x_2^4$

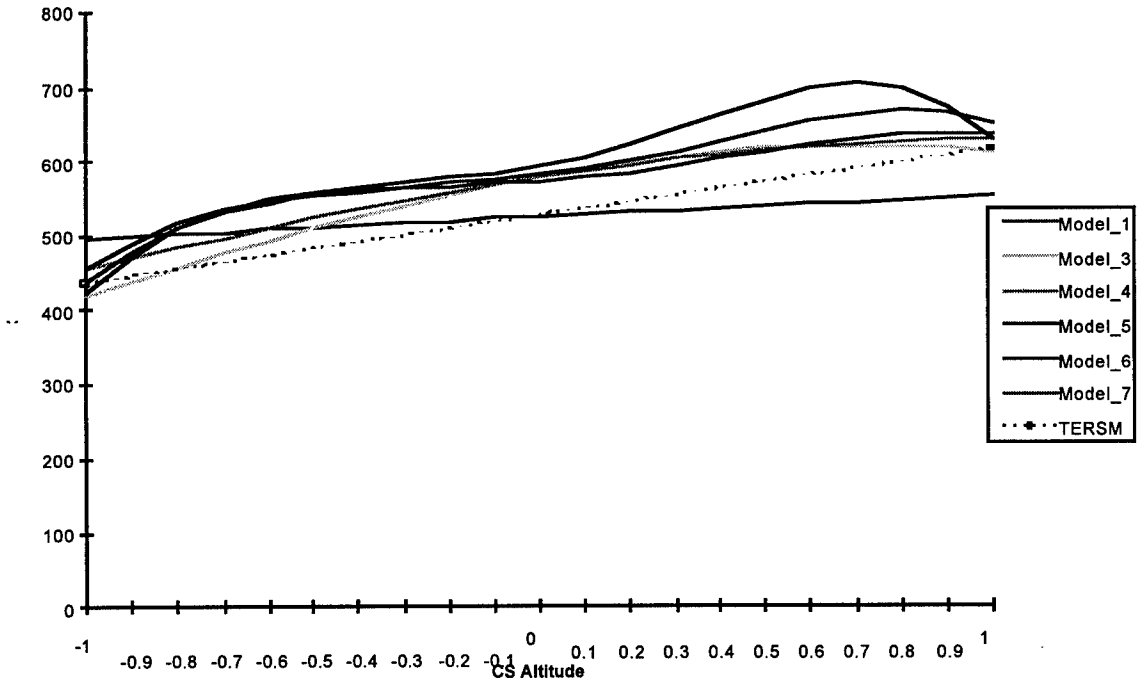
Table E-3, TERSM Data Points and Metamodel Outputs

TERSM Data Points and Metamodel Outputs										
Alt	**Vel**	**Az**	**CC**	**Mdl_1**	**Mdl_2**	**Mdl_3**	**Mdl_4**	**Mdl_5**	**Mdl_6**	**Mdl_7**
1.0	1.0	1.0	1.0	553	628	613	629	635	629	650
1.0	1.0	1.0	-1.0	225	209	192	203	200	195	193
1.0	1.0	-1.0	1.0	395	341	342	341	346	339	342
1.0	1.0	-1.0	-1.0	67	51	58	56	38	56	56
1.0	-1.0	1.0	1.0	245	283	260	248	271	247	248
1.0	-1.0	1.0	-1.0	106	53	73	70	63	72	71
1.0	-1.0	-1.0	1.0	171	129	114	126	116	113	118
1.0	-1.0	-1.0	-1.0	32	28	47	36	34	49	41
-1.0	1.0	1.0	1.0	497	467	419	454	454	424	437
-1.0	1.0	1.0	-1.0	280	209	213	210	217	219	217
-1.0	1.0	-1.0	1.0	340	343	338	333	333	328	334
-1.0	1.0	-1.0	-1.0	123	114	137	128	119	134	123
-1.0	-1.0	1.0	1.0	189	171	172	184	179	172	183
-1.0	-1.0	1.0	-1.0	161	102	115	104	91	111	101
-1.0	-1.0	-1.0	1.0	116	82	95	88	92	99	97
-1.0	-1.0	-1.0	-1.0	88	42	64	67	26	64	58
0.0	0.0	0.0	0.0	224	533	570	544	550	572	555
-1.0	0.0	0.0	0.0	224	490	523	505	509	523	511
1.0	0.0	0.0	0.0	224	514	476	505	533	476	515
0.0	-1.0	0.0	0.0	138	228	191	212	247	191	218
0.0	1.0	0.0	0.0	310	412	439	423	431	439	430
0.0	0.0	-1.0	0.0	166	385	371	423	464	372	430
0.0	0.0	1.0	0.0	282	508	683	608	590	685	616
0.0	0.0	0.0	-1.0	135	430	249	301	361	251	306
0.0	0.0	0.0	1.0	313	635	633	572	566	637	584
0.5	0.5	0.5	0.5	364	620	781	667	627	625	625
0.5	0.5	0.5	-0.5	238	467	445	458	414	414	418
0.5	0.5	-0.5	0.5	296	524	567	537	530	535	523
0.5	-0.5	0.5	0.5	244	491	501	485	648	679	651
0.5	0.5	-0.5	-0.5	170	397	299	356	342	327	343
0.5	-0.5	0.5	0.5	244	491	501	485	648	679	651
0.5	-0.5	0.5	-0.5	165	386	299	346	487	476	488

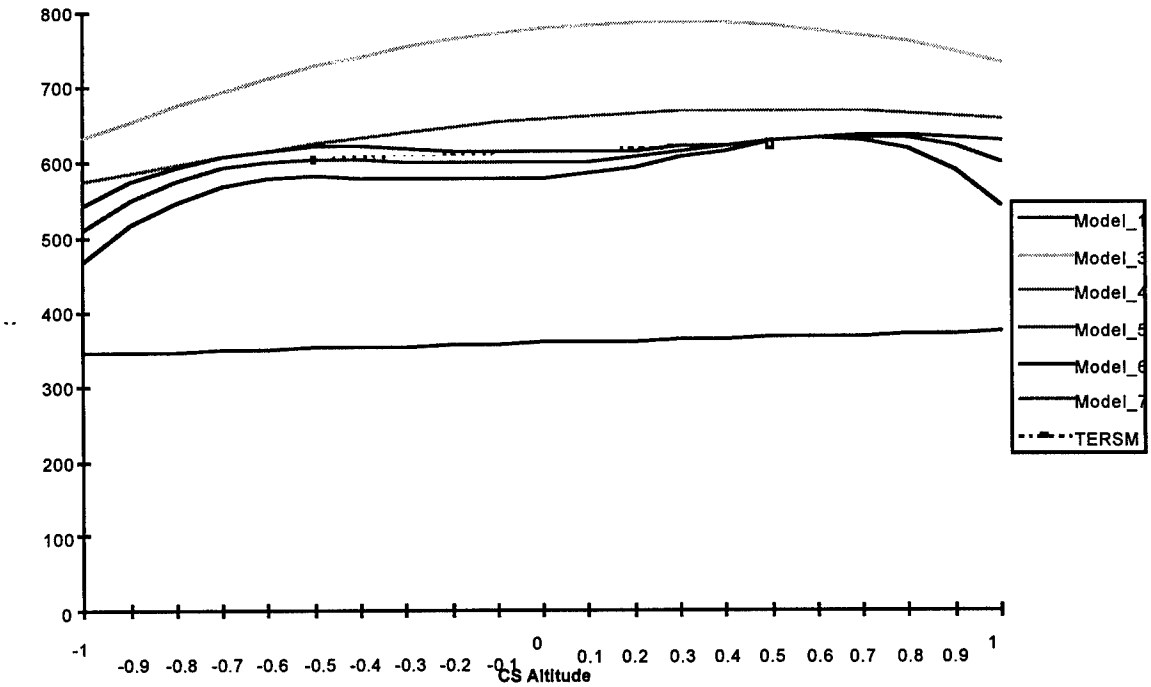
TERSM Data Points and Metamodel Outputs										
Alt	**Vel**	**Az**	**CC**	**Mdl_1**	**Mdl_2**	**Mdl_3**	**Mdl_4**	**Mdl_5**	**Mdl_6**	**Mdl_7**
0.5	-0.5	-0.5	0.5	197	422	357	394	578	583	569
0.5	-0.5	-0.5	-0.5	118	342	218	283	442	417	437
-0.5	0.5	0.5	0.5	350	580	726	624	618	581	603
-0.5	0.5	0.5	-0.5	252	461	485	465	443	450	449
-0.5	0.5	-0.5	0.5	282	512	580	530	550	546	539
-0.5	-0.5	0.5	0.5	231	457	463	457	650	635	644
-0.5	-0.5	0.5	-0.5	179	386	332	360	517	526	525
-0.5	-0.5	-0.5	0.5	183	404	349	380	597	574	582
-0.5	-0.5	-0.5	-0.5	132	346	250	305	476	474	480
-0.5	0.0	0.0	0.0	224	519	565	534	570	610	583
0.5	0.0	0.0	0.0	224	531	539	534	549	582	559
0.0	-0.5	0.0	0.0	181	434	391	415	567	584	571
0.0	0.5	0.0	0.0	267	525	593	546	506	515	507
0.0	0.0	-0.5	0.0	195	480	474	489	513	519	512
0.0	0.0	0.5	0.0	253	542	644	584	575	593	581
0.0	0.0	0.0	-0.5	180	481	412	441	448	447	450
0.0	0.0	0.0	0.5	269	584	658	590	609	613	603

There are twenty graphs included in this appendix, organized as four sets. Within each set, one of the four input parameters is varied, while constant values are used for the other three parameters. There five graphs within each set, representing five different combinations of constant values for the remaining three parameters. Each graph contains curves for six of the metamodels (Metamodel 2 was not used) and the actual data points from the TERSM simulation runs. A dotted line is used to highlight the TERSM data points; it should not be interpreted as a linear interpolation of the data.

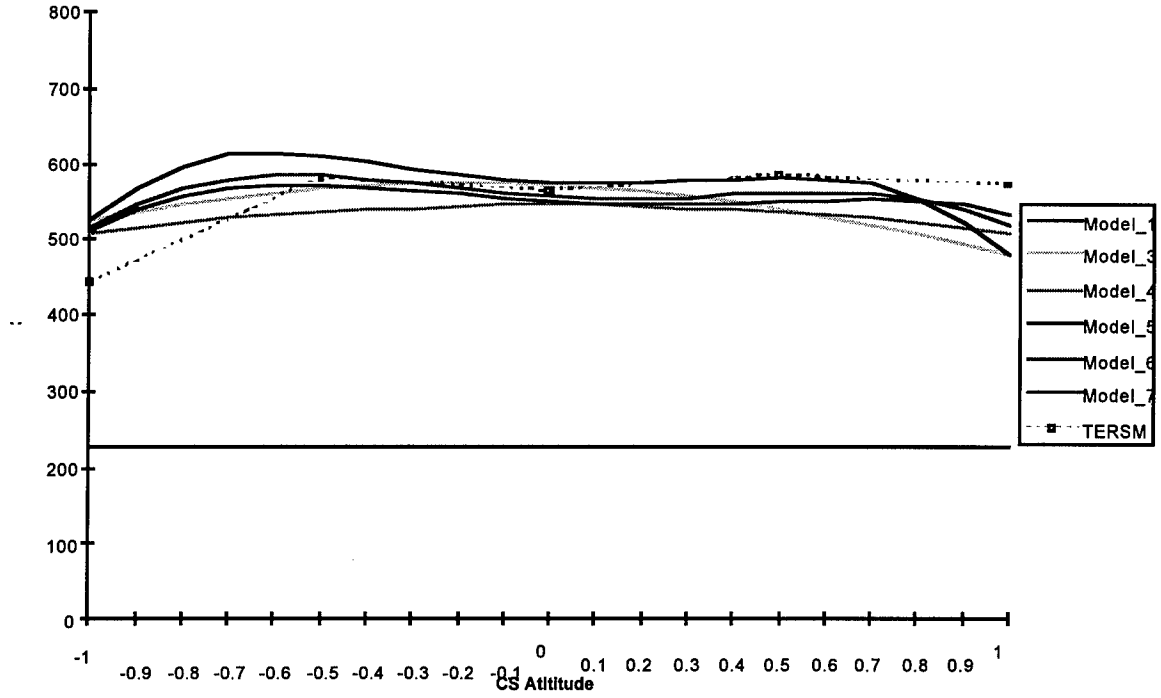
Vel, Az, ChanCap = (1,1,1)



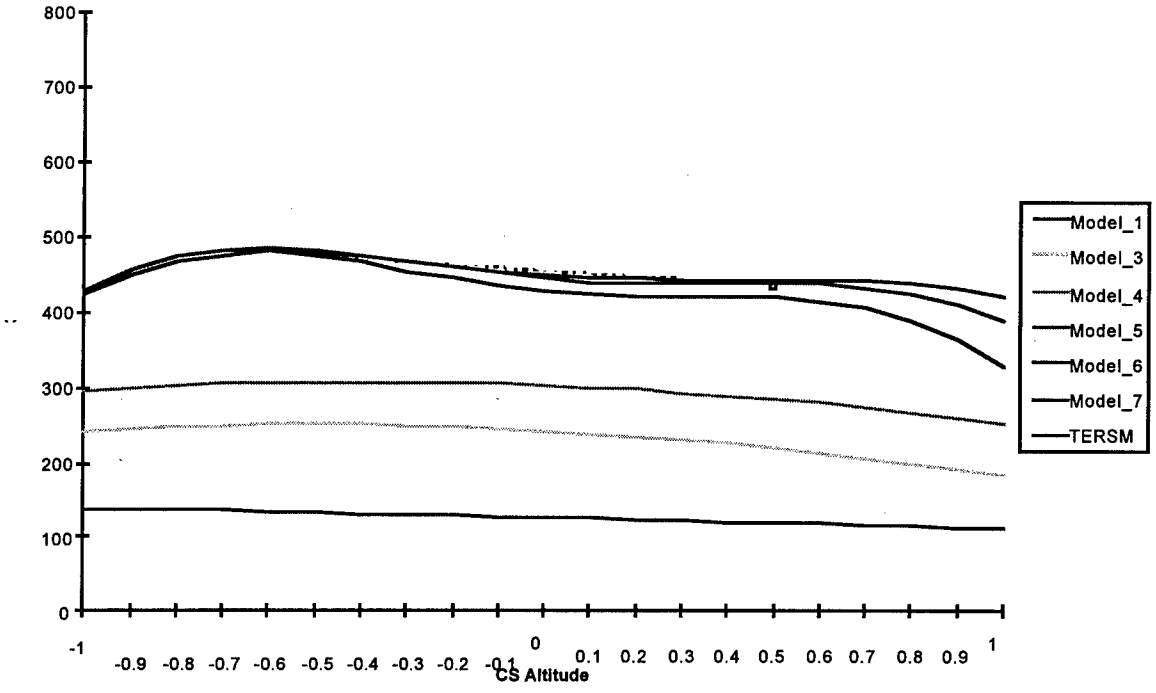
Vel, Az, ChanCap = (.5,.5,.5)

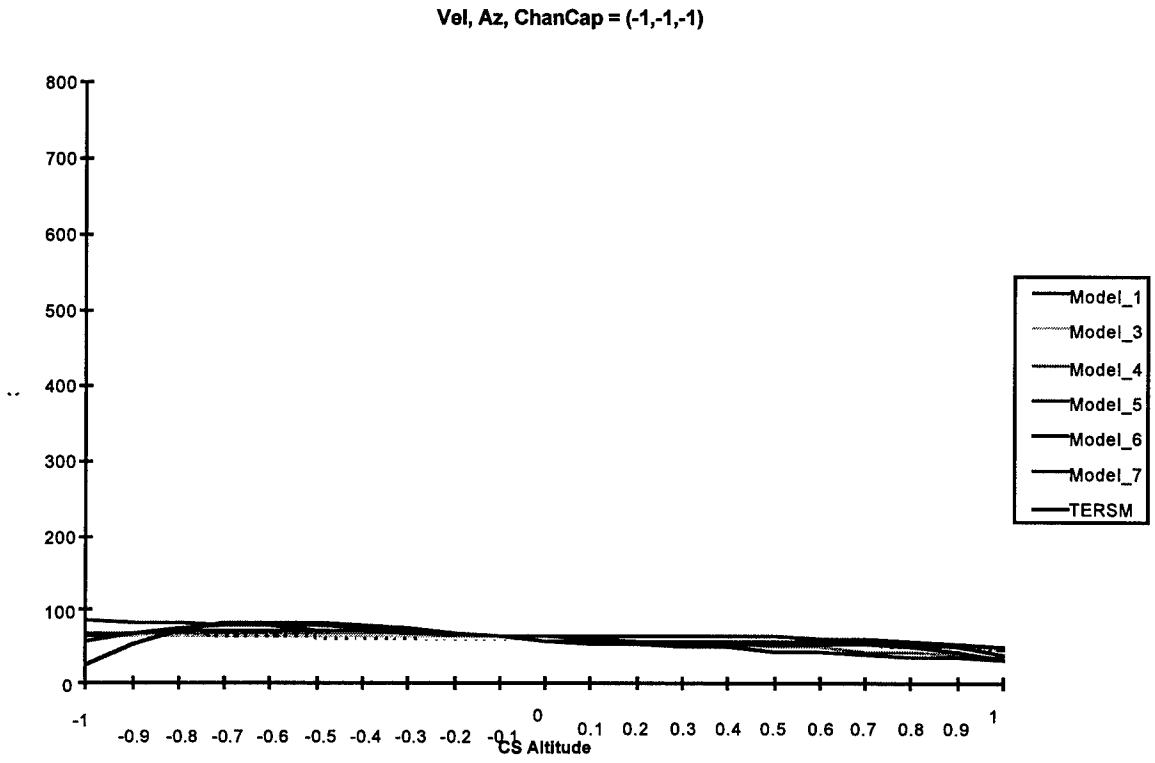


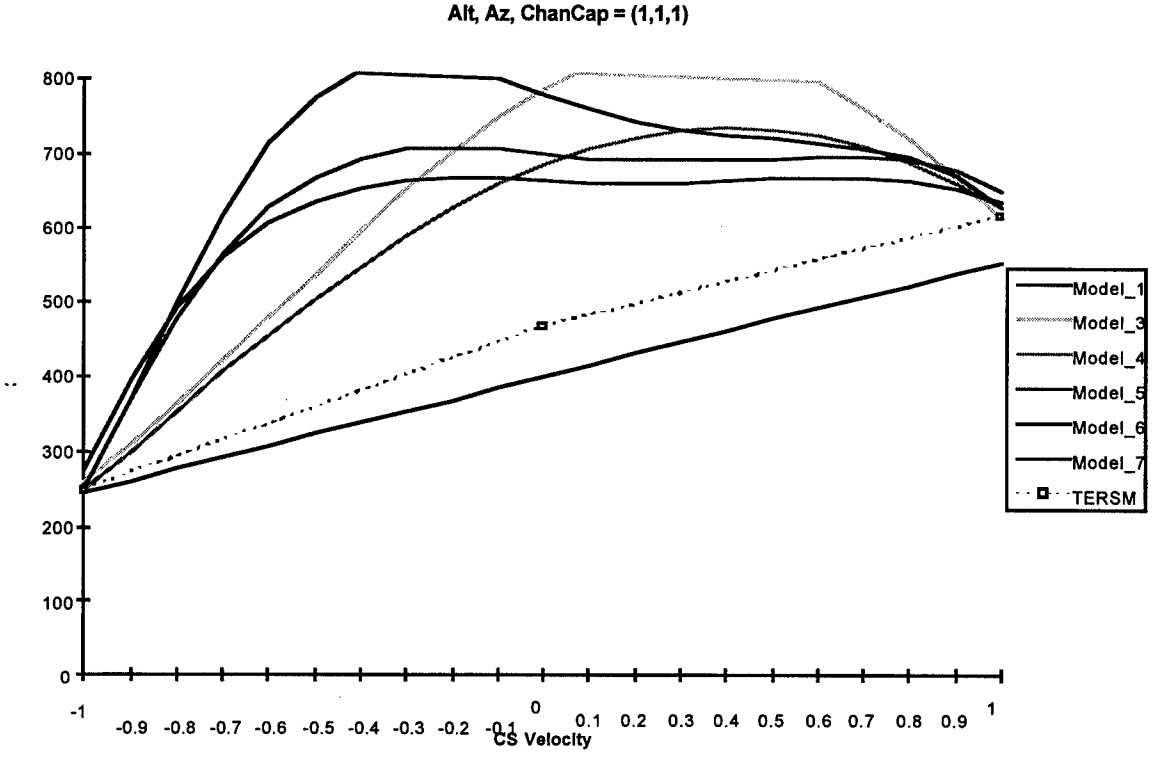
Vel, Az, ChanCap = (0,0,0)

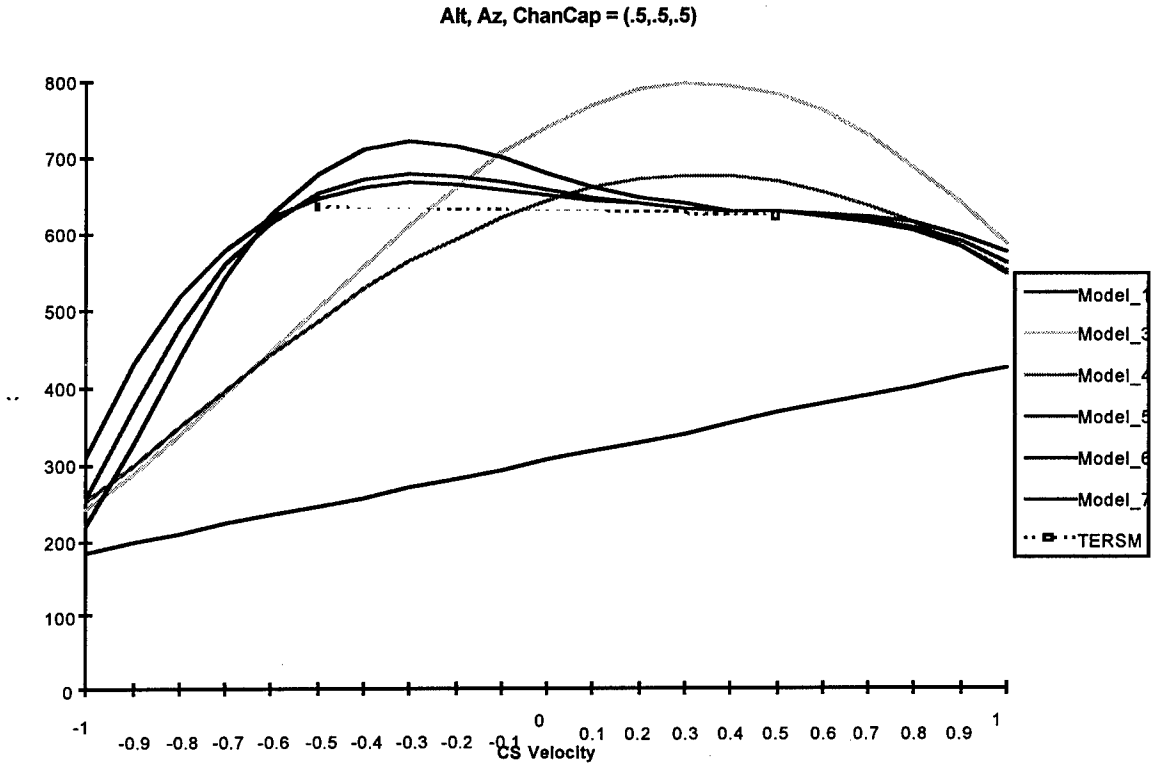


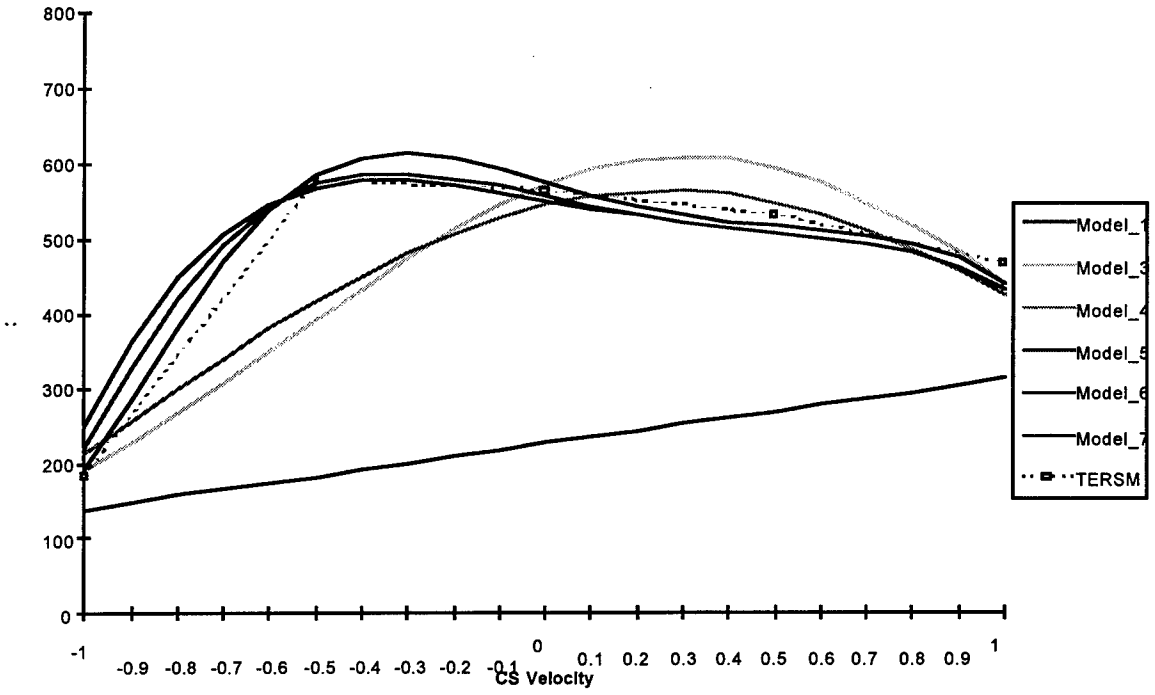
Vel, Az, ChanCap = (-.5,-.5,-.5)



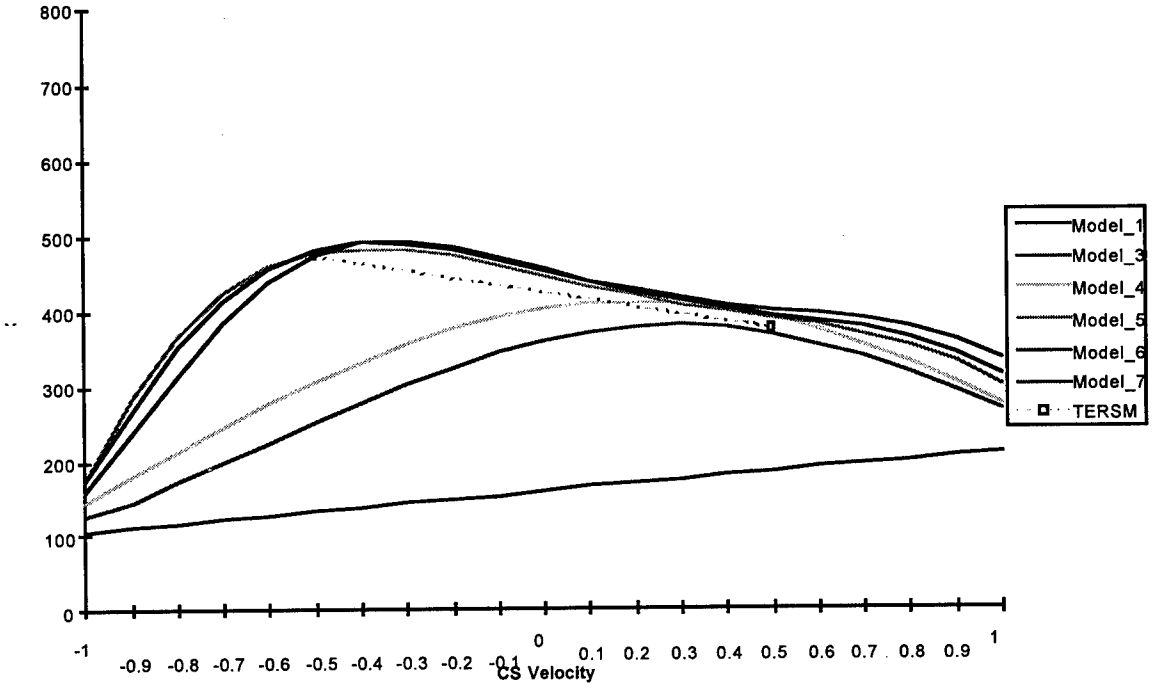

Vel, Az, ChanCap = (-1,-1,-1)
800
700
600
500
400
300
200
100
0
-1
-0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1
0
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
1
CS Altitude
Model_1
Model_3
Model_4
Model_5
Model_6
Model_7
TERSM


Alt, Az, ChanCap = (1,1,1)
800
700
600
500
400
300
200
100
0
-1
-0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1
0
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
1
CS Velocity
Model_1
Model_3
Model_4
Model_5
Model_6
Model_7
TERSM

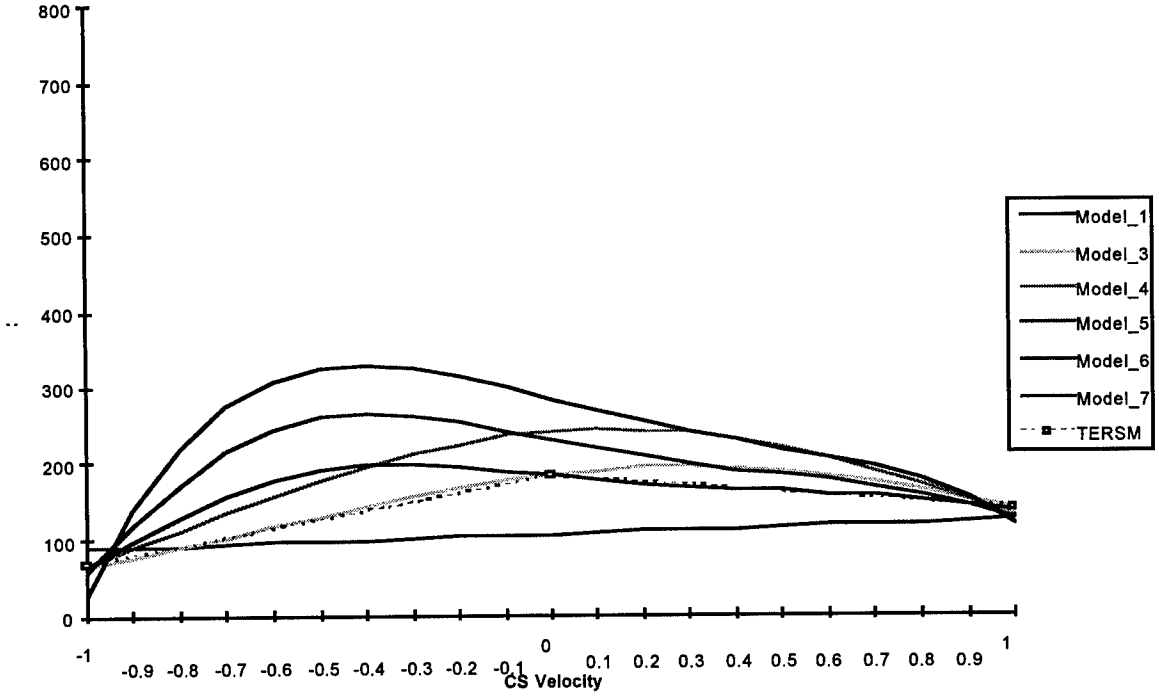

Alt, Az, ChanCap = (.5,.5,.5)
800
700
600
500
400
300
200
100
0
-1
-0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1
0
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
1
CS Velocity
Model_1
Model_3
Model_4
Model_5
Model_6
Model_7
TERSM


Alt, Az, ChanCap = (0,0,0)
800
700
600
500
400
300
200
100
0
-1
-0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1
0
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
1
CS Velocity
Model_1
Model_3
Model_4
Model_5
Model_6
Model_7
TERSM

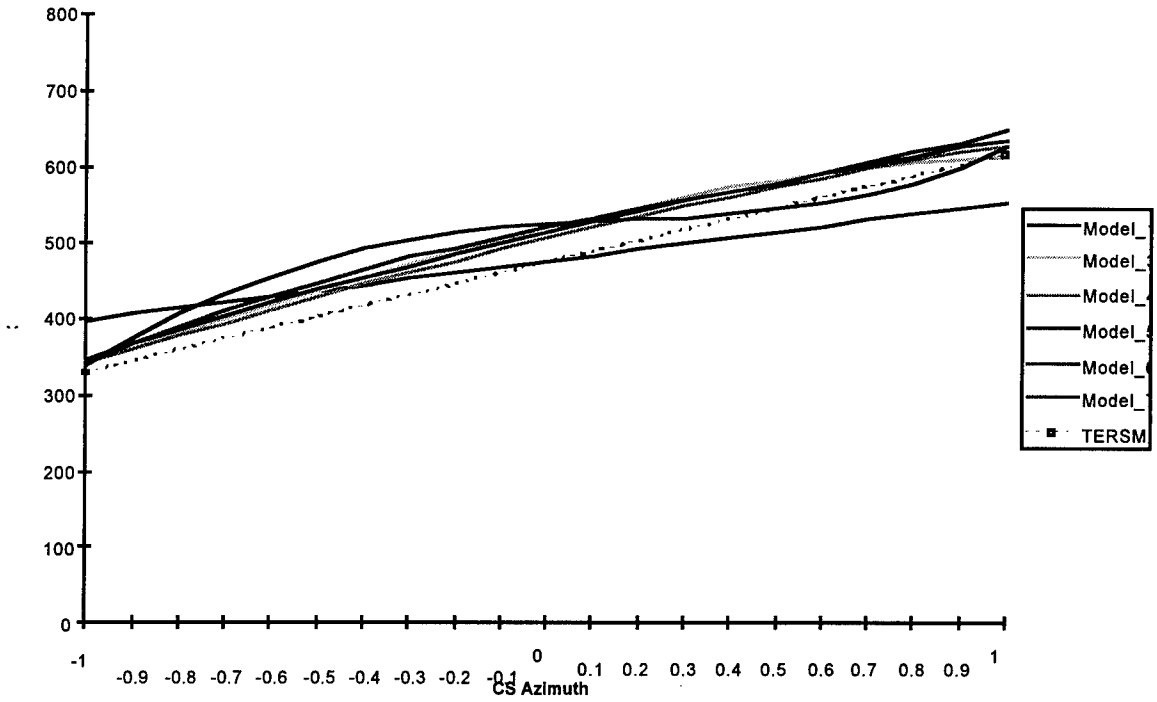
Alt, Az, ChanCap = (-.5,-.5,-.5)



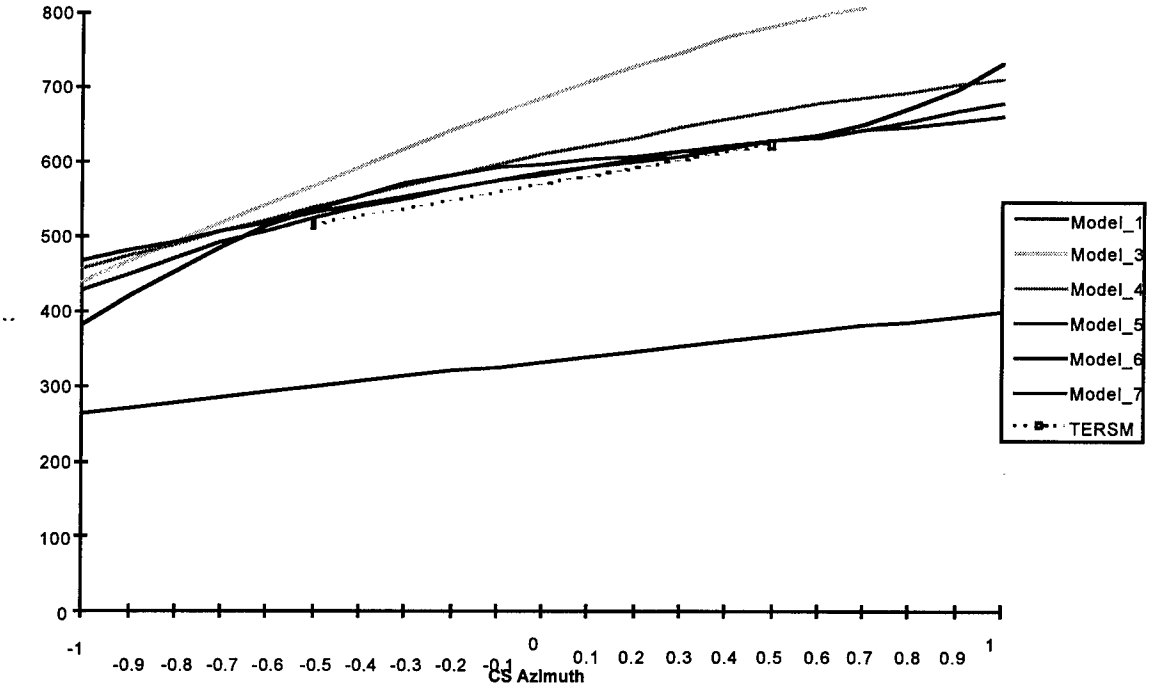
Alt, Az, ChanCap = (-1,-1,-1)



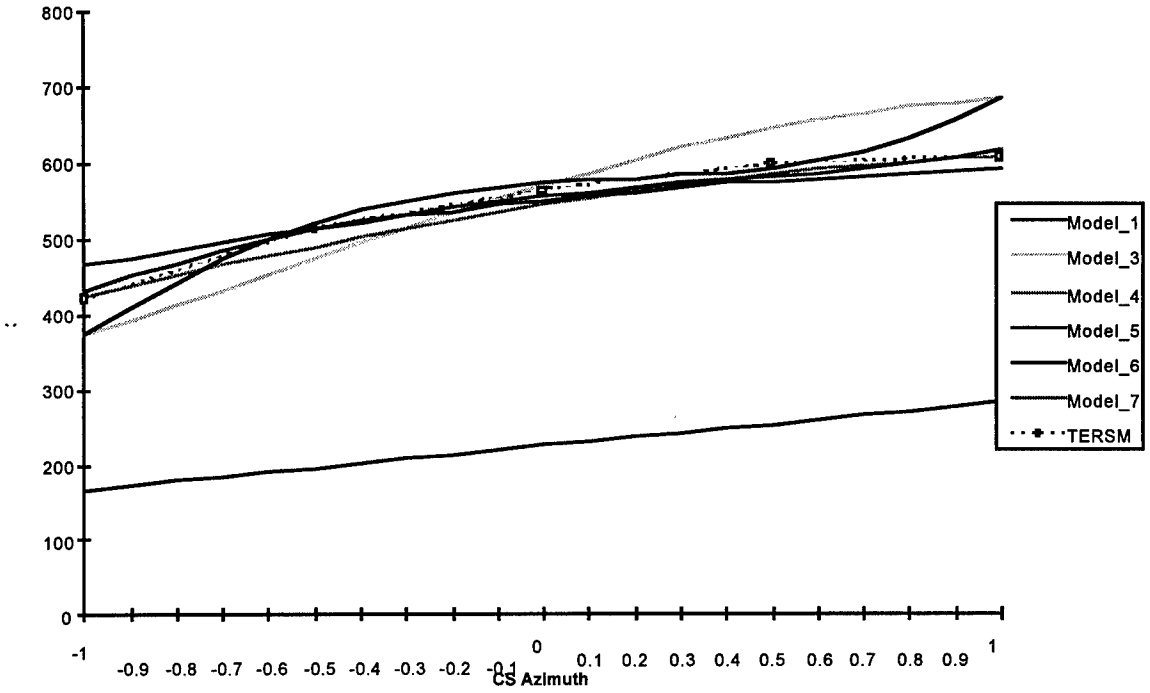
Alt, Vel, ChanCap = (1,1,1)



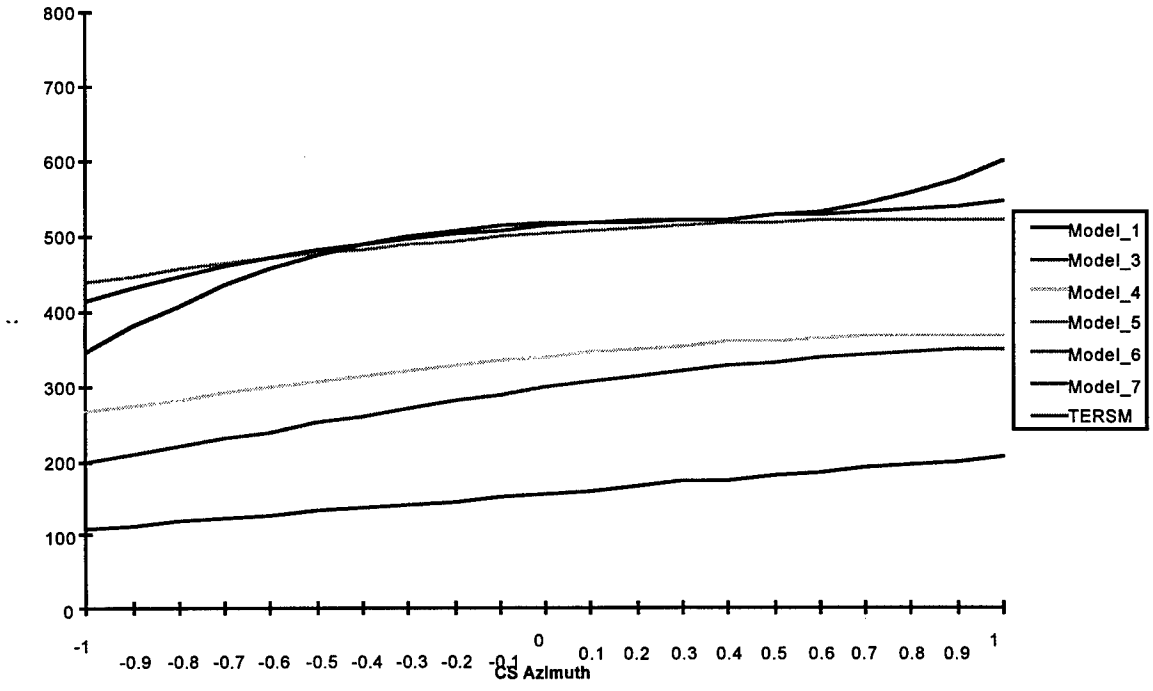
Alt, Vel, ChanCap = (.5,.5,.5)



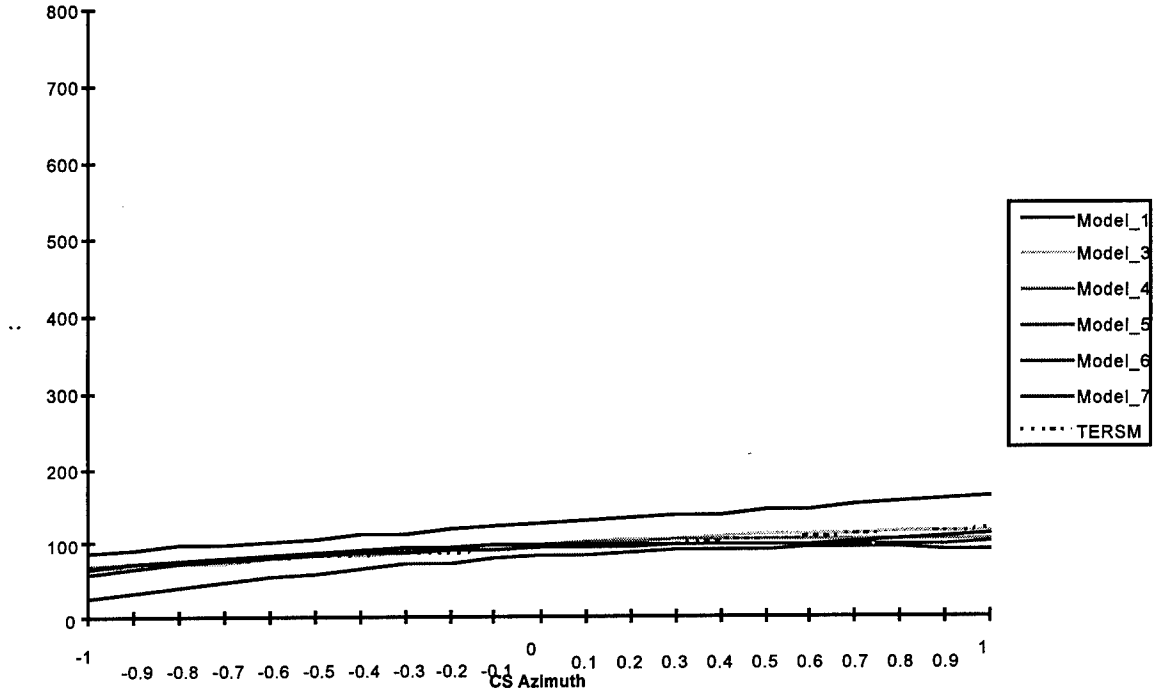
Alt, Vel, ChanCap = (0,0,0)



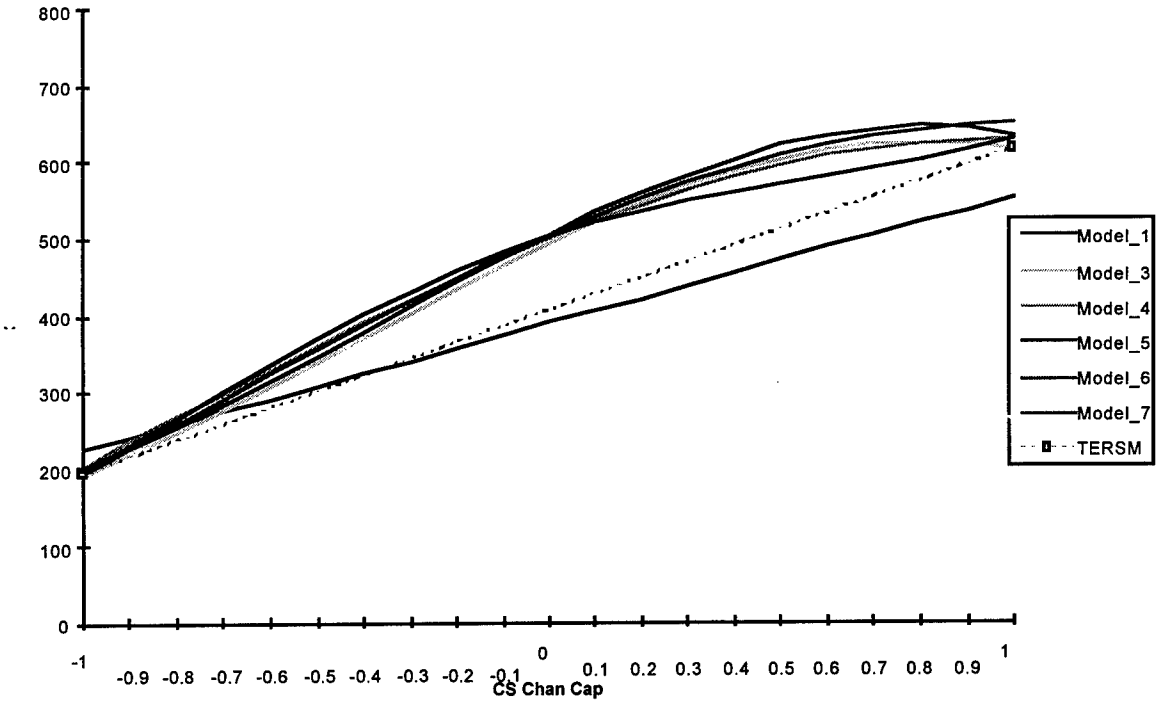
Alt, Vel, ChanCap = (-5.,-.5,-.5)



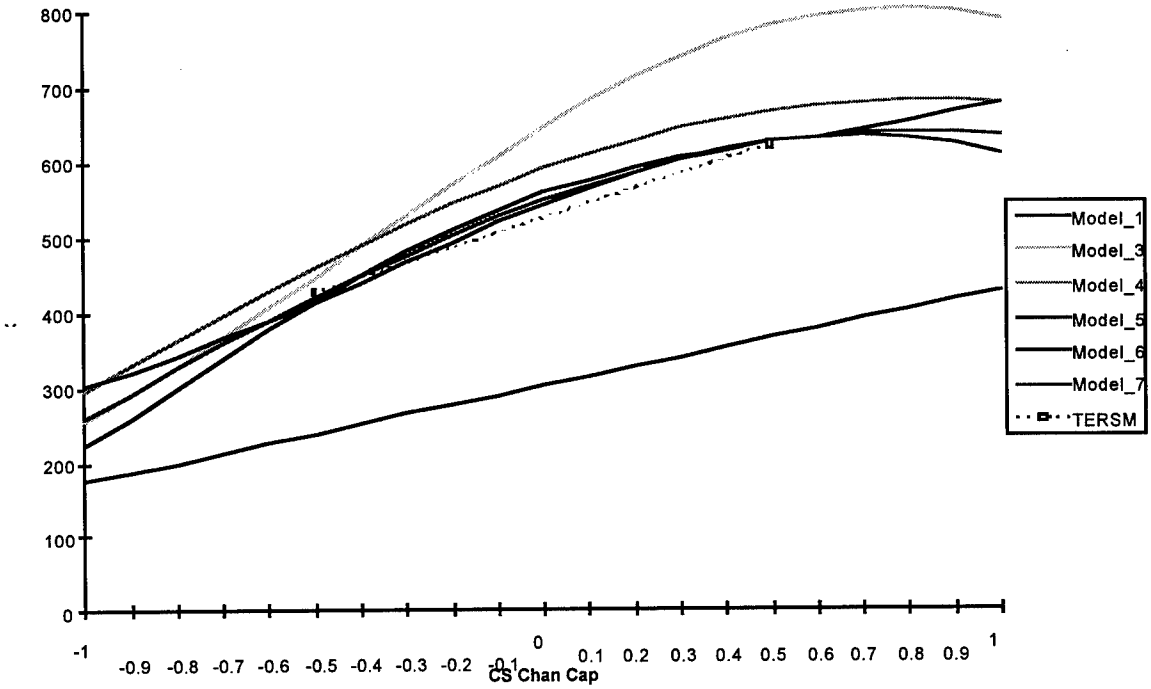
Alt, Vel, ChanCap = (-1,-1,-1)



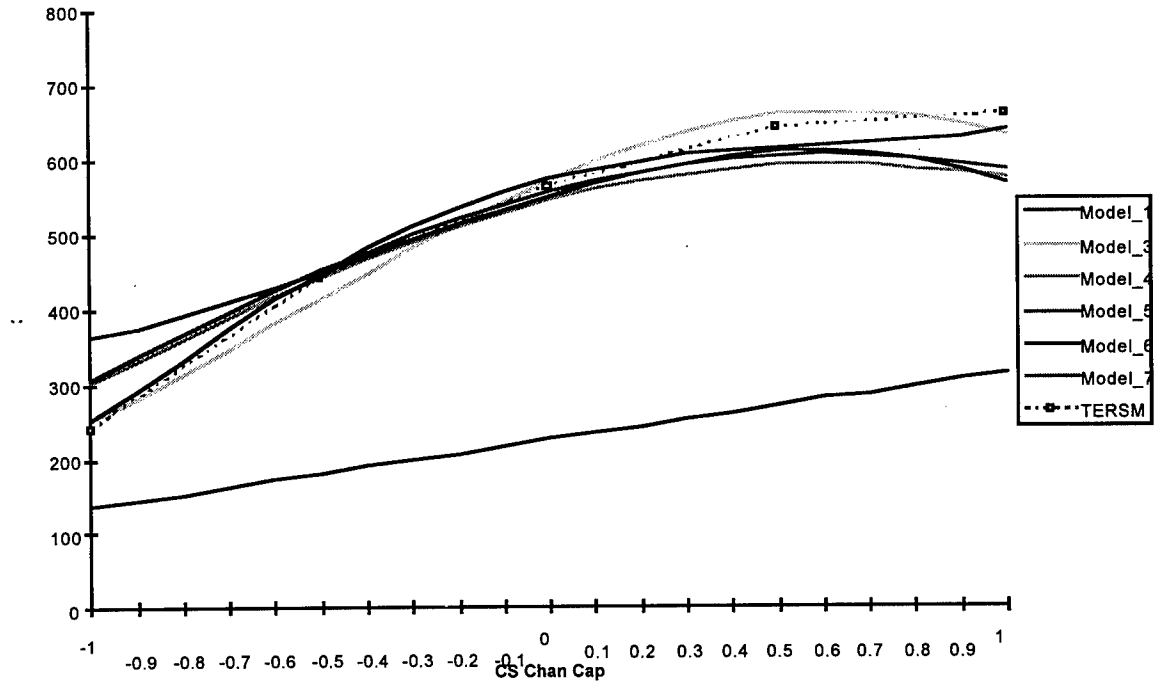
Alt, Vel, Az = (1,1,1)



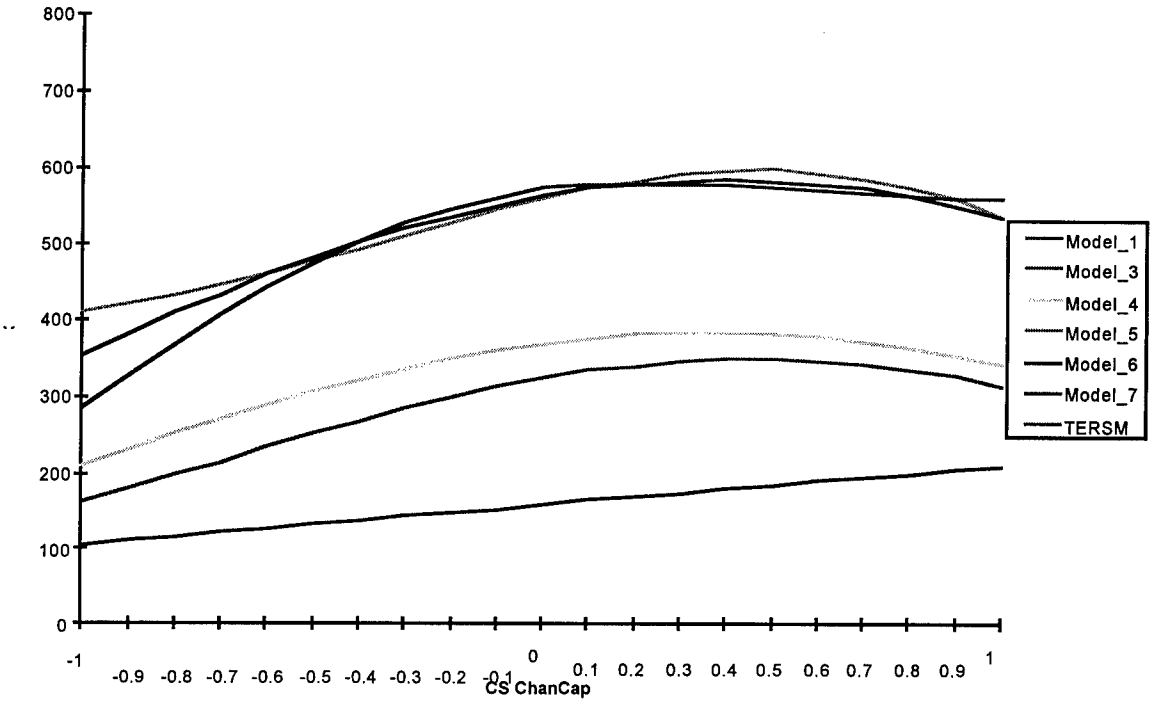
Alt, Vel, Az = (.5,.5,.5)



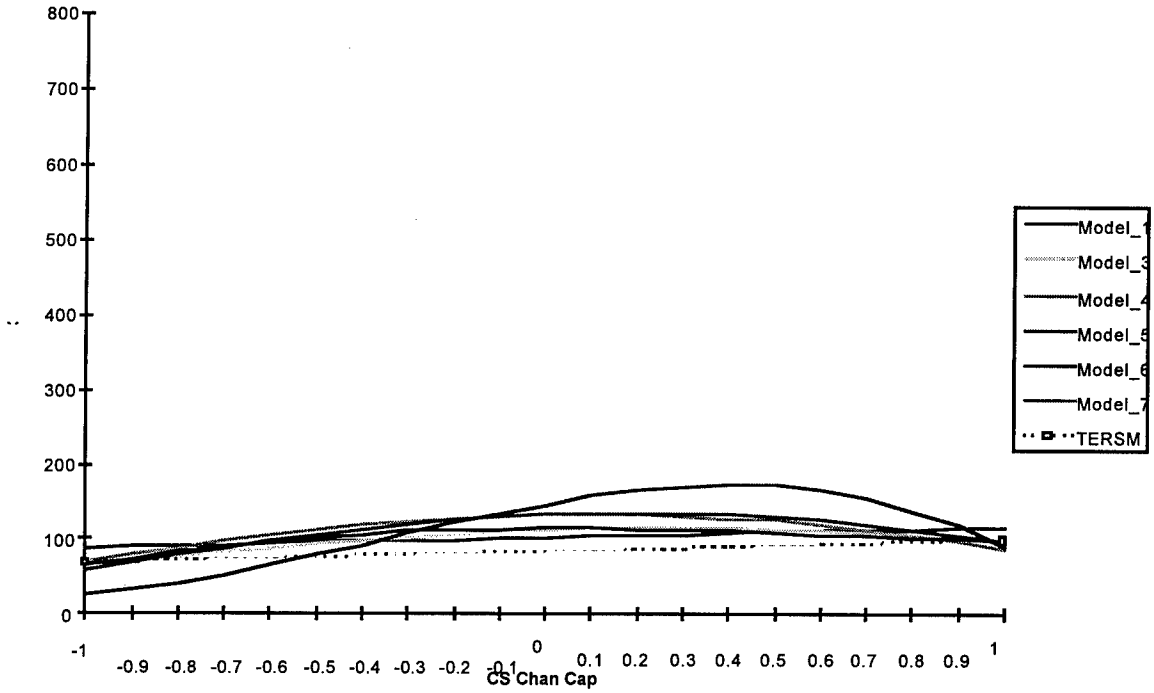
Alt, Vel, Az = (0,0,0)



Alt, Vel, Az = (-.5,-.5,-.5)



Alt, Vel, Az = (-1,-1,-1)



F. Mapping a Discrete Event Model into a Qualitative Model

The purpose of this appendix is to explore this idea of synergistic use of qualitative models for identifying abstractions. A comprehensive discussion of qualitative models can be found in (Kuipers 1994), while Section 4 provides a brief introduction to key qualitative modeling concepts. We take a simple discrete event model originally described by Fishwick and Zeigler (1992), and define a comparable qualitative model.

This appendix is divided into two sections. The first section includes a description of how the qualitative model was constructed. The second section summarizes our observations as a result of the analysis documented in Section F.1.

F.1 Building a Qualitative Model

The first step in analyzing the synergistic application of qualitative modeling is to construct a qualitative model which can be compared to a discrete event model of the same functional process. We chose a model of boiling water described by Fishwick and Zeigler (1992). This model was chose for several reasons:

- Since the model is of a physical process, it fits the domain most often represented in qualitative models.
- The model is simple (only a few hundred lines of code), but completely describes a physical process.
- The model was used in a discussion of multimodeling, and provides a useful example for comparison.

In this section, we walk through the process used to create the qualitative version of the model.

The physical system being modeled is described by Fishwick and Zeigler (1992) as follows. A pot of boiling water is on a stovetop electric heating element. Initially, the pot is filled to some predetermined level with water. A small amount of detergent is added to simulate the foaming activity that occurs naturally when boiling certain foods. This system has one input or control, the temperature knob. The knob is considered to be in one of two states: on or off (on is 190 degrees Centigrade; off is α, the ambient temperature). The model is based on the following assumptions:

- The input (knob turning) can change at any time. The input trajectories are piecewise continuous with two possible values (ON, OFF).
- The liquid level (height) does not increase until the liquid starts to boil.
- When the liquid starts to boil, a layer of foam increases in height until it either overflows the pot or the knob is turned off.
- The liquid level decreases during the boiling and overflow phases only.

We begin by identifying the quantity spaces for the variables of the system:

Temperature	T	α ... 100 ... ∞
Height of the water	H_w	0 ... H_t
Height of the foam	H_f	0 ... H_t

where

α is the ambient room temperature

H_t is the height of the top of the container of water.

We know that there are certain constraints on the relationship of variables describing the system:

$H_f \geq H_w$

In addition to these three state parameters, there is a fourth parameter, the switch, designated I (input). The switch has two values (ON, OFF). The only external events to this system are changes to I.

In constructing the qualitative model, we first define the initial state in which I=OFF. The qualitative state variables are defined as follows:

T	$\langle \alpha, std \rangle$
H_w	$\langle (0,H_t), std \rangle$
H_f	$\langle H_w, std \rangle$

Since all variables are in a steady state, this is a quiescent state, in which there is no successor state without an external event. In other words, the pot sitting on the heating element will not change "spontaneously". This is the COLD (M_1) state in Fishwick and Zeigler.

Now consider the situation caused by turning on the heating element (I = ON). The temperature increases, although until the water reaches the boiling point, the height of the water and foam do not change. This state, identified as HEATING in Fishwick and Zeigler, is represented as follows:

T	$\langle (\alpha,100), inc \rangle$
H_w	$\langle (0,H_t), std \rangle$
H_f	$\langle H_w, std \rangle \rangle$

Without external intervention, there is only one successor state:

T	$\langle 100, std \rangle$
H_w	$\langle (0,H_t), dec \rangle$
H_f	$\langle (H_w,H_t), inc \rangle$

This is the BOILING state defined by Fishwick and Zeigler.

At this point, there are two possible successor states, depending on which of the following situations occurs first: the foam overflows the container, or the liquid boils dry. The state in which the foam overflows the container (OVERFLOW in Fishwick and Zeigler) is expressed as follows:

T $\langle 100, std \rangle$
H_w $\langle (0,H_t), dec \rangle$
H_f $\langle H_t, std \rangle$

while the second state (UNDERFLOW in Fishwick and Zeigler) is defined as:

T $\langle (100,\infty), inc \rangle$
H_w $\langle 0, std \rangle$
H_f $\langle 0, std \rangle$

According to the description of system behavior in Fishwick and Zeigler, the underflow state is also a successor of the overflow state. However, even this simple qualitative rendering of the physical system illustrates some problems with the stated assumptions. H_f is discontinuous in the transition from OVERFLOW to UNDERFLOW, going from H_t to 0. An additional underflow state could be defined based on an assumption that the foam gradually drops after the water boils away:

T $\langle (100,\infty), inc \rangle$
H_w $\langle 0, std \rangle$
H_f $\langle (0,H_t), dec \rangle$

The previously defined underflow state becomes the successor state for this state, reached when the foam is completely gone.

Note also that the assumption that H_f increases during boiling assumes that the foam increases at a faster rate than the water decreases. This assumption could be retracted by using the amount of foam (A_f) rather than the height of the foam. The parameters are related as follows:

$$H_f = H_w + A_f$$

The state definitions are given below (using the Fishwick and Zeigler state names):

COLD		HEATING		BOILING		OVERFLOW	
T	$\langle \alpha, std \rangle$	T	$\langle (\alpha,100), inc \rangle$	T	$\langle 100, std \rangle$	T	$\langle 100, std \rangle$
H_w	$\langle (0,Ht), std \rangle$	H_w	$\langle (0,H_t), std \rangle$	H_w	$\langle (0,H_t), dec \rangle$	H_w	$\langle (0,H_t), dec \rangle$
A_f	$\langle 0, std \rangle$	A_f	$\langle 0, std \rangle$	A_f	$\langle (0,_{Ht}), inc \rangle$	Af	$\langle (0,H_t), inc$

ADD'L UNDERFLOW		UNDERFLOW	
T	$\langle(100,\infty), inc\rangle$	T	$\langle(100,\infty), inc\rangle$
H_w	$\langle 0, std\rangle$	H_w	$\langle 0, std\rangle$
A_f	$\langle(0,H_t), dec\rangle$	A_f	$\langle 0, std\rangle$

Along with the definition of a constraint $H_w + A_f \leq H_t$, this set of qualitative state definitions allows the increase of the amount of foam to be less than the decrease in water height, such that the overall height of the foam can decrease while the water is boiling. Note that we in this particular case we have expanded the number of physical systems represented by the model by retracting an assumption without changing the number of qualitative states used to represent system behavior. (The additional complexity of this model is due to the requirement to check the above constraint.)

We now look at expanding the model to include the effects of turning the heating element off. The physical manifestation of this event is that the temperature of the water drops, the water stops boiling, and the foam gradually dissipates. In terms of our qualitative model, we can express this situation (referred to as COOLING) in terms of qualitative state variables as follows:

T	$\langle(\alpha,100), dec\rangle$
H_w	$\langle(0,H_t), std\rangle$
H_f	$\langle(H_w,H_t) dec\rangle$

Eventually, assuming no external inputs (i.e. the heat is not turned back on), the system will eventually return to its initial quiescent state. However, since there are two state variables which are moving back to their steady state condition, the initial COLD state is not a direct successor of this state. Thus we must define two additional qualitative states, either of which can be the successor state to the COOLING states defined above.

ALMOST-COLD-1		ALMOST-COLD-2	
T	$\langle\alpha, std\rangle$	T	$\langle(\alpha,100), dec\rangle$
H_w	$\langle(0,H_t), std\rangle$	H_w	$\langle(0,H_t), std\rangle$
H_f	$\langle(H_w,H_t) dec\rangle$	H_f	$\langle H_w, std\rangle$

The state ALMOST-COLD-1 represents the state in which the temperature has fallen back to the ambient room temperature before the foam has dissipated. ALMOST-COLD-2 is the opposite situation. The initial COLD state is the successor state for both of these states. Note that the qualitative model abstracts the fact that the level of water in the pot after boiling and cooling is not the same as the starting level. Since the qualitative state variable only reflects that there is some water in the pot, expressed as $H_w \in (0,H_t)$.

Note that the definition of the COOLING state assumes that the model is in the BOILING or OVERFLOW states when the I is set to OFF; otherwise, there is no foam, and it cannot be decreasing. If the system is in the HEATING state, there is no foam, and the transition due to

turning off the heat is to the ALMOST-COLD-2 state defined above. If the system is in the UNDERFLOW state, the effect of turning off the heat is to transition to a new state defined as:

COOLING EMPTY 1

T	$\langle(100,\infty), dec\rangle$
H_w	$\langle 0, std\rangle$
H_f	$\langle 0, std\rangle$

The successor states are a sequence of states as the temperature falls past the boiling point to the ambient room temperature. These states are defined as:

COOLING EMPTY 2

T	$\langle 100, dec\rangle$
H_w	$\langle 0, std\rangle$
H_f	$\langle 0, std\rangle$

COOLING EMPTY 3

T	$\langle(\alpha,100), dec\rangle$
H_w	$\langle 0, std\rangle$
H_f	$\langle 0, std\rangle$

Similarly, additional states must be defined for the case when I is set to OFF when the system is in the ADDITIONAL UNDERFLOW state. The initial transition is to:

COOLING WITH FOAM 1

T	$\langle(100,\infty), dec\rangle$
H_w	$\langle 0, std\rangle$
H_f	$\langle(0,H_t), dec\rangle$

Depending on which state value reached a landmark first, successor states could be those described above, or the following:

COOLING WITH FOAM 2

T	$\langle 100, dec\rangle$
H_w	$\langle 0, std\rangle$
H_f	$\langle(0,H_t), std\rangle$

COOLING WITH FOAM 3

T	$\langle(\alpha,100), dec\rangle$
H_w	$\langle 0, std\rangle$
H_f	$\langle(0,H_t), std\rangle$

One final assumption is added to simplify this discussion: the knob is not turned back on after the water has boiled dry (i.e., while in the COOLING EMPTY states or the COOLING WITH FOAM states). Additional states with increasing temperature but no water would be required to represent such situations, and practically such a situation should not occur under normal circumstances. These states should be added for a complete model, but will be ignored for the purposes of this illustration of a qualitative model.

The state transition diagram for this model is shown in Figure F-1. States are shown as nodes in the graph, with transitions depicted as arrows. Transitions triggered by external events are shown as dotted lines and gray arrowheads, while solid lines with solid black arrowsheads represent transitions which occur due to internal events.

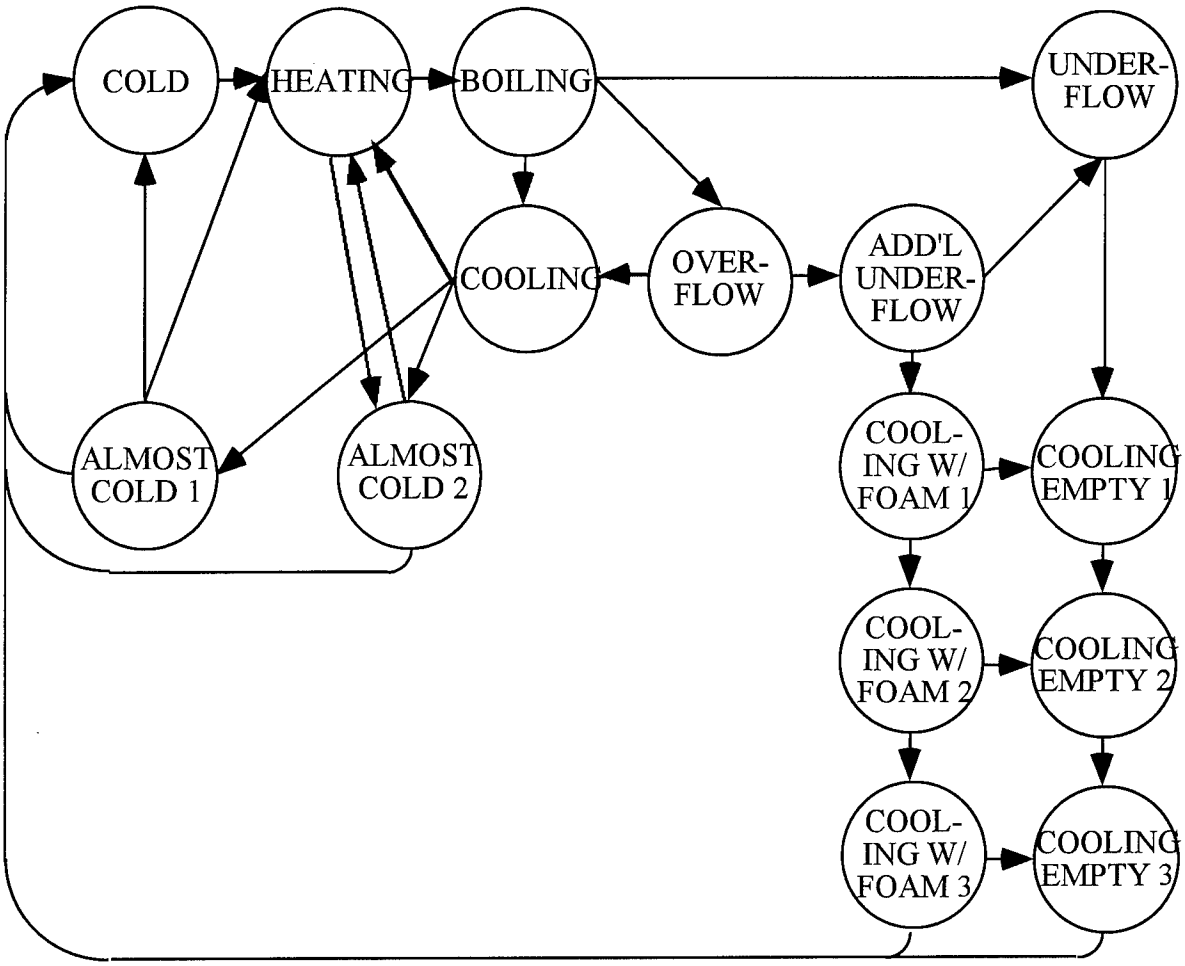


Figure F-1, Boiling Water Qualitative Model State Transition Diagram

F.2 Observations

While this exercise of qualitative state definition provides insight into the model itself, building an actual qualitative simulation model turns out to result in a fairly trivial model that does not provide much useful insight into the concepts of qualitative models as a surrogate abstraction domain. The quantity spaces can be represented in QSIM syntax as follows:

```
(quantity spaces
     (temp      (alpha    100   tmax))
     (Hwater    (0    Htop))
     (Hfoam     (0    Htop))
```

System state is thus based on the state of three variables, and their interactions are substantially different in different model regions. The COLD state is a quiescent state, in that there are no state changes unless the system is acted upon by an external event (turning on the heat). At this point there is a region change, to represent the system HEATING. There are no significant constraints in the model for this region, as the height of the water and the height of

the foam remain constant until the temperature reaches 100. To generate successor states, we again need to add a transition, this time to a region representing the behavior of the system as the water boils. In summary, the state transitions originally defined in Fishwick and Zeigler (1992) end up as separate regions within the qualitative model, with trivial processing within each region.

A more challenging problem would provide more insight into the concept of qualitative models as surrogate abstraction domains. Thus the next step in following this research thread is to apply this methodology to a more complex problem, to determine whether the approach scales up to non-trivial cases.

F.3 References

Fishwick, P. and B. Zeigler. 1992. A multimodel methodology for qualitative model engineering. *ACM Transactions on Modeling and Computer Simulation* 2:52-81.

Kuipers, B. 1994. *Qualitative reasoning: modeling and simulation with incomplete knowledge.* MIT Press, Cambridge, Massachusetts.

G. Software Documentation

The focus of this research effort was on conceptual development, rather than software development. However, some programs were written to investigate key concepts. The purpose of this appendix is to document these software components. Section G.1 includes notes for the program that was written to generate TERSM metamodel output shown in Appendix E. Because this work builds on the preceding TERSM studies, this program is documented in detail. The second software component is an algorithm for launch point calculation in RWWM Subroutine EX45 with varying iteration intervals.

G.1 TERSM Metamodel Data Generation Software Notes

The TERSM Metamodel Data Generation program was written to generate metamodel outputs for the TERSM metamodels described in Zeimer et. al. (1993). This appendix provides documentation of that program. There are seven subsections in this appendix, which provide information on various aspects of the software, as follows:

1. Software requirements;
2. Software design;
3. Software test requirements;
4. Software test results;
5. Software user's notes;
6. Software programmering notes; and the
7. Program listing.

G.1.1 Requirements Notes

Objective: The TERSM Metamodel (TERSM_MM) program will provide the user with a simple mechanism for generating the data for the TERSM metamodels described in Zeimer et. al. (1993). These metamodels are of the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_{12} x_1 x_2 + \ldots$$

that is, the sum of a series of terms which are the products of a coefficient and one or more of four inputs representing altitude, velocity, azimuth, and channel capacity.

Input Requirements: The inputs to the program include:

Model coefficients (the β terms in the equations): there is one set of coefficients for each metamodel, with more than one metamodel. The number of metamodels shall be a compilable parameter. The coefficients can be any real value in the range - ∞, ∞. Model coefficients shall be read in from a file.

Parameter values: One value for each of the four parameters for each test case. The parameter values have been centered and scaled, so they are all in the range [-1, 1]. Parameter values shall either be read in from a file or automatically generated by the program based on user specifications. The user shall be able to specify either of two cases for automatically generating test case parameter values:

1. A sequence of test cases in which a single parameter is varied by a fixed user specified increment, and the other three parameters are assigned user-specified constant values; and
2. A sequence of test cases in which two parameters are varied by (separate) fixed increments, and the other two parameters are assigned user-specified constant values; in this case the user must also specify the metamodel to be used to calculate the data.

File names can be specified in the command line; if not supplied there, the program must prompt for missing file names.

Output Requirements: The program shall generate the results of the metamodel in a text file formatted such that it ca be read in by a Microsoft Excel program. Excel shall be used to generate graphs of the data. One format of the output shall be as follows:

x_1 x_2 x_3 x_4 m_1 m_2 ....

where x_i is the value of the test case parameter, and m_i is the output of the i^{th} metamodel. A second format shall be output which shows a two-dimensional response surface. The file format shall be an array with rows representing various values of one of the test case parameters, and columns representing various values of a second parameter. Values generated by a single metamodel shall be output for each set of test cases generated in this format.

The program shall also output the metamodel coefficients in the metamodel equation form shown above in the Objective paragraph.

Functional Requirements: The program shall prompt the user for necessary inputs, then calculate the metamodel output and write the results to a file. All inputs shall be checked to ensure that they are valid.

G.1.2 Design Notes

This program will be written as a single module. Separate procedures will be written to handle the initialization of the coefficients, the test case parameters, and the output file. High level pseudocode for the main program is listed below:

```
initialize_coefficients
initialize_test_case_parameters
initialize_output_file
if output based on single varying parameter
    then generate data for each metamodel for each test case
```

```
        else generate data for each parameter pair.

    initialize_coefficients
        if file not named in command line
            then prompt user for name;
        open file and read in coefficients.

    initialize_test_case_parameters
        if file named in command line
            then open file
            else
                case
                metamodel coefficient output: print_coeffs
                file option: prompt user for name and open file and read in data
                single varying param: get parameter, increment, and other values from
                    user, then generate test cases
                dual varying param: get parameters and increments, constant parameter
                    values, and metamodel, then generate test cases.

    initialize_output_file
        if file not named in command line
            then prompt user for name;
        open file.

    print_coeffs
        for each metamodel
            for each term
                if coefficient ≠ 0
                    then output coefficient and term.
```

The code for printing out the metamodel equation will be placed in a separate procedure, largely because it includes many lines of repetitive code for each coefficient.

G.1.3 Test Description Notes

Test the following cases:

1. All files specified in the command line.
2. No files specified in the command line, but all data in files.
3. Output metamodel coefficients in equation form.
4. Automatically generate single varying parameter test cases.
5. Automatically generate dual varying parameter test cases.
6. Quit from test case generation without generating output.
7. Generate data for TERSM test points.

Test the following error conditions.

8. Files specified in command line do not exist.
9. Files specified by user in response to prompt do not exist.
10. Number of metamodel cofficient sets in the coefficients file does not match the number of models permitted in program.
11. Number of test cases specified in the test case parameter file exceeds amount of storage allocated for test case parameters.
12. User reponse to prompt is not a valid option.
    a. Invalid test case option
    b. Test case parameter value out of range.
    c. Increment value for automatically generated test cases is out of range.
    d. Number of increments in single varying parameter case requires more test cases than accommodated by test case parameter storage.
    e. Number of increments in dual varying parameter case requires more test cases than accommodated by test case parameter storage.

G.1.4 Test Results

The following script shows the results of running the tests listed above. Titles were added to this script to assist in mapping the portion of the output to the individual tests. Some error messages were modified after test completion to provide a uniform error message format. Those changes are reflected in the code listing in Section G.1.7.

TEST 1: All files specified in the command line.

```
userve% cat new_coeffs.in
7
224.118 0.0 85.75 57.75 89.0 0.0 0.0 27.75 21.0 47.25 0.0 0.0 0.0 0.0 0.0
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
532.633 11.944 91.944 61.778 102.333 0.0 16.0 27.75 21.0 47.25 19.75 12.25 0.0 12.5 0.0
     0.0 -30.608 -212.608 -86.108 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
6.346 -.047 .417 .306 .467 -.025 .075 .17 0.0 .098 -.028 .066 0.0 0.0 -.078
     -.049 -.133 -.679 -.125 -.363 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
23.319 0.0 2.994 2.042 3.288 0.0 .488 1.006 .407 1.155 .257 .4 0.0 0.0 -.288
     0.0 -.841 -5.757 -.701 -2.683 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
549.756 -32.722 -111.537 62.778 180.556 0.0 16.441 26.647 20.882  46.971 18.676 12.508
9.785 12.892 0.0
     0.0 61.972 0.0 -22.962 -86.491 44.667 203.481 0.0 -78.222 -90.454 -210.918 0.0 0.0
6.35 -0.047 -.308 .075 .267 -.022 .073 .162 .021 .099 -.029 .066 .014 0.0 -.078
     -.049 .26 0.0 -.125 -.359 0.0 .725 .23 .199 -.398 -.682 0.0 0.0
```

```
23.567 -.669 -2.842 1.298 3.344 0.0 .491 .963 .414 1.155 .231 .404 .198 .201 -.285
     0.0 2.037 0.0 -.788 -2.743 .714 5.836 .744 0.0 -2.947 -5.823 0.000 0.0
userve% cat testx000.in
21
1.0 0.0 0.0 0.0
0.9 0.0 0.0 0.0
0.8 0.0 0.0 0.0
0.7 0.0 0.0 0.0
0.6 0.0 0.0 0.0
0.5 0.0 0.0 0.0
0.4 0.0 0.0 0.0
0.3 0.0 0.0 0.0
0.2 0.0 0.0 0.0
0.1 0.0 0.0 0.0
0.0 0.0 0.0 0.0
-0.1 0.0 0.0 0.0
-0.2 0.0 0.0 0.0
-0.3 0.0 0.0 0.0
-0.4 0.0 0.0 0.0
-0.5 0.0 0.0 0.0
-0.6 0.0 0.0 0.0
-0.7 0.0 0.0 0.0
-0.8 0.0 0.0 0.0
-0.9 0.0 0.0 0.0
-1.0 0.0 0.0 0.0
userve% tersm_mm new_coeffs.in testx000.in test.out
userve% cat test.out
CS_Alt CS_Vel CS_Az CS_CC Model_1 Model_2 Model_3 Model_4 Model_5 Model_6 Model_7
1.000000 0.000000 0.000000 0.000000  224  514  476  505  533  476  515
0.900000 0.000000 0.000000 0.000000  224  519  491  512  544  522  538
0.800000 0.000000 0.000000 0.000000  224  523  504  519  549  553  552
0.700000 0.000000 0.000000 0.000000  224  526  517  525  551  572  559
0.600000 0.000000 0.000000 0.000000  224  529  528  530  550  580  560
0.500000 0.000000 0.000000 0.000000  224  531  539  534  549  582  559
0.400000 0.000000 0.000000 0.000000  224  533  548  538  547  580  557
0.300000 0.000000 0.000000 0.000000  224  533  556  540  546  576  554
0.200000 0.000000 0.000000 0.000000  224  534  562  542  546  573  553
0.100000 0.000000 0.000000 0.000000  224  534  567  543  547  571  553
0.000000 0.000000 0.000000 0.000000  224  533  570  544  550  572  555
-0.100000 0.000000 0.000000 0.000000  224  531  572  543  554  577  559
-0.200000 0.000000 0.000000 0.000000  224  529  573  542  558  584  565
-0.300000 0.000000 0.000000 0.000000  224  526  571  540  563  592  572
-0.400000 0.000000 0.000000 0.000000  224  523  569  538  568  602  578
-0.500000 0.000000 0.000000 0.000000  224  519  565  534  570  610  583
-0.600000 0.000000 0.000000 0.000000  224  514  559  530  570  614  584
-0.700000 0.000000 0.000000 0.000000  224  509  552  525  566  611  580
-0.800000 0.000000 0.000000 0.000000  224  503  544  519  556  596  568
-0.900000 0.000000 0.000000 0.000000  224  497  534  512  537  568  546
-1.000000 0.000000 0.000000 0.000000  224  490  523  505  509  523  511
```

TEST 2: No files specified in command line, but all data in files.

```
userve% tersm_mm
Enter Metamodel Coefficient file name: new_coeffs.in
Enter test case data option (? lists options):F
Enter test case file name: testx000.in
Enter output file name: test.out
userve% cat test.out
CS_Alt CS_Vel CS_Az CS_CC Model_1 Model_2 Model_3 Model_4 Model_5 Model_6 Model_7
1.000000 0.000000 0.000000 0.000000  224  514  476  505  533  476  515
0.900000 0.000000 0.000000 0.000000  224  519  491  512  544  522  538
0.800000 0.000000 0.000000 0.000000  224  523  504  519  549  553  552
0.700000 0.000000 0.000000 0.000000  224  526  517  525  551  572  559
0.600000 0.000000 0.000000 0.000000  224  529  528  530  550  580  560
0.500000 0.000000 0.000000 0.000000  224  531  539  534  549  582  559
0.400000 0.000000 0.000000 0.000000  224  533  548  538  547  580  557
0.300000 0.000000 0.000000 0.000000  224  533  556  540  546  576  554
0.200000 0.000000 0.000000 0.000000  224  534  562  542  546  573  553
0.100000 0.000000 0.000000 0.000000  224  534  567  543  547  571  553
0.000000 0.000000 0.000000 0.000000  224  533  570  544  550  572  555
-0.100000 0.000000 0.000000 0.000000  224  531  572  543  554  577  559
-0.200000 0.000000 0.000000 0.000000  224  529  573  542  558  584  565
-0.300000 0.000000 0.000000 0.000000  224  526  571  540  563  592  572
-0.400000 0.000000 0.000000 0.000000  224  523  569  538  568  602  578
-0.500000 0.000000 0.000000 0.000000  224  519  565  534  570  610  583
-0.600000 0.000000 0.000000 0.000000  224  514  559  530  570  614  584
-0.700000 0.000000 0.000000 0.000000  224  509  552  525  566  611  580
-0.800000 0.000000 0.000000 0.000000  224  503  544  519  556  596  568
-0.900000 0.000000 0.000000 0.000000  224  497  534  512  537  568  546
-1.000000 0.000000 0.000000 0.000000  224  490  523  505  509  523  511
```

TEST 3: Output metamodel coefficients in equation form.

```
userve% tersm_mm
Enter Metamodel Coefficient file name: new_coeffs.in
Enter test case data option (? lists options):C
Model Number 1
224.118 + 85.750x2 + 57.750x3 + 89.000x4
+ 27.750x1x4 + 21.000x2x3 + 47.250x2x4

Model Number 2
532.633 + 11.944x1 + 91.944x2 + 61.778x3 + 102.333x4
+ 16.000x1x3 + 27.750x1x4 + 21.000x2x3 + 47.250x2x4 + 19.750x3x4
+ 12.250x1x2x3 + 12.500x1x3x4
-30.608x1**2  -212.608x2**2  -86.108x3**2

Model Number 3
6.346 -0.047x1 + 0.417x2 + 0.306x3 + 0.467x4
-0.025x1x2 + 0.075x1x3 + 0.170x1x4 + 0.098x2x4 -0.028x3x4
```

```
+  0.066x1x2x3  -0.078x2x3x4
-0.049x1x2x3x4  -0.133x1**2  -0.679x2**2  -0.125x3**2  -0.363x4**2

Model Number 4
23.319 + 2.994x2 + 2.042x3 + 3.288x4
+ 0.488x1x3 + 1.006x1x4 + 0.407x2x3 + 1.155x2x4 + 0.257x3x4
+  0.400x1x2x3  -0.288x2x3x4
-0.841x1**2  -5.757x2**2  -0.701x3**2  -2.683x4**2

Model Number 5
549.756 -32.722x1 -111.537x2 + 62.778x3 + 180.556x4
+ 16.441x1x3 + 26.647x1x4 + 20.882x2x3 + 46.971x2x4 + 18.676x3x4
+ 12.508x1x2x3 + 9.785x1x2x4 + 12.892x1x3x4
+  61.972x1**2  -22.962x3**2  -86.491x4**2
+  44.667x1**3  +  203.481x2**3  -78.222x4**3
-90.454x1**4  -210.918x2**4


Model Number 6
6.350 -0.047x1 -0.308x2 + 0.075x3 + 0.267x4
-0.022x1x2 + 0.073x1x3 + 0.162x1x4 + 0.021x2x3 + 0.099x2x4 -0.029x3x4
+ 0.066x1x2x3 + 0.014x1x2x4 -0.078x2x3x4
-0.049x1x2x3x4  +  0.260x1**2  -0.125x3**2  -0.359x4**2
+ 0.725x2**3 + 0.230x3**3 + 0.199x4**3
-0.398x1**4  -0.682x2**4

Model Number 7
23.567 -0.669x1 -2.842x2 + 1.298x3 + 3.344x4
+ 0.491x1x3 + 0.963x1x4 + 0.414x2x3 + 1.155x2x4 + 0.231x3x4
+ 0.404x1x2x3 + 0.198x1x2x4 + 0.201x1x3x4 -0.285x2x3x4
+  2.037x1**2  -0.788x3**2  -2.743x4**2
+ 0.714x1**3 + 5.836x2**3 + 0.744x3**3
-2.947x1**4  -5.823x2**4
```

TEST 4: Automatically generate single varying parameter test case.

```
userve% tersm_mm new_coeffs.in
Enter test case data option (? lists options):1
Input variable to vary (? to list options): A
Enter increment for altitude: .1
Input centered, scaled velocity: [-1,1]:0
Input centered, scaled azimuth: [-1,1]:0
Input centered, scaled chan cap.: [-1,1]:0
Enter output file name: test.out
userve% cat test.out
CS_Alt CS_Vel CS_Az CS_CC Model_1 Model_2 Model_3 Model_4 Model_5 Model_6 Model_7
-1.000000 0.000000 0.000000 0.000000  224  490  523  505  509  523  511
-0.900000 0.000000 0.000000 0.000000  224  497  534  512  537  568  546
-0.800000 0.000000 0.000000 0.000000  224  503  544  519  556  596  568
```

```
-0.700000 0.000000 0.000000 0.000000  224  509  552  525  566  611  580
-0.600000 0.000000 0.000000 0.000000  224  514  559  530  570  614  584
-0.500000 0.000000 0.000000 0.000000  224  519  565  534  570  610  583
-0.400000 0.000000 0.000000 0.000000  224  523  569  538  568  602  578
-0.300000 0.000000 0.000000 0.000000  224  526  571  540  563  592  572
-0.200000 0.000000 0.000000 0.000000  224  529  573  542  558  584  565
-0.100000 0.000000 0.000000 0.000000  224  531  572  543  554  577  559
0.000000 0.000000 0.000000 0.000000   224  533  570  544  550  572  555
0.100000 0.000000 0.000000 0.000000   224  534  567  543  547  571  553
0.200000 0.000000 0.000000 0.000000   224  534  562  542  546  573  553
0.300000 0.000000 0.000000 0.000000   224  533  556  540  546  576  554
0.400000 0.000000 0.000000 0.000000   224  533  548  538  547  580  557
0.500000 0.000000 0.000000 0.000000   224  531  539  534  549  582  559
0.600000 0.000000 0.000000 0.000000   224  529  528  530  550  580  560
0.700000 0.000000 0.000000 0.000000   224  526  517  525  551  572  559
0.800000 0.000000 0.000000 0.000000   224  523  504  519  549  553  552
0.900000 0.000000 0.000000 0.000000   224  519  491  512  544  522  538
1.000000 0.000000 0.000000 0.000000   224  514  476  505  533  476  515
```

TEST 5: Automatically generate dual varying parameter test cases.

```
userve% !t
tersm_mm new_coeffs.in
Enter test case data option (? lists options):2
Input two variables to vary (? to list options): AV
Enter increment for altitude: .1
Enter increment for velocity: .5
Input centered, scaled azimuth: [-1,1]:0
Input centered, scaled chan cap.: [-1,1]:0
Which metamodel should be used (1-7)? 7
Enter output file name: test2.out
userve% cat test2.out
 -1.00 -0.50 0.00 0.50 1.00
-1.00  190  526  511  465  391
-0.90  212  561  546  498  422
-0.80  225  584  568  519  441
-0.70  233  596  580  530  452
-0.60  236  600  584  534  455
-0.50  235  599  583  533  454
-0.40  232  594  578  528  450
-0.30  228  587  572  522  444
-0.20  224  581  565  516  439
-0.10  220  575  559  511  434
0.00  218  571  555  507  430
0.10  216  569  553  505  428
0.20  216  569  553  504  428
0.30  217  570  554  506  429
```

```
0.40  218  572  557  508  431
0.50  220  575  559  510  433
0.60  221  576  560  511  434
0.70  220  574  559  510  433
0.80  215  567  552  503  427
0.90  207  554  538  490  415
1.00  193  530  515  469  395
```

TEST 6: Quit from test case generation without generating output.

```
userve% rm test.out
userve% !t
tersm_mm new_coeffs.in
Enter test case data option (? lists options):Q
userve% cat test.out
cat: cannot open test.out
```

TEST 7: Generate data for TERSM test points.

```
userve% cat datapoints.in
49
1 1 1 1
1 1 1 -1
1 1 -1 1
1 1 -1 -1
1 -1 1 1
1 -1 1 -1
1 -1 -1 1
1 -1 -1 -1
-1 1 1 1
-1 1 1 -1
-1 1 -1 1
-1 1 -1 -1
-1 -1 1 1
-1 -1 1 -1
-1 -1 -1 1
-1 -1 -1 -1
0 0 0 0
-1 0 0 0
1 0 0 0
0 -1 0 0
0 1 0 0
0 0 -1 0
0 0 1 0
0 0 0 -1
0 0 0 1
.5 .5 .5 .5
.5 .5 .5 -.5
```

```
.5 .5 -.5 .5
.5 -.5 .5 .5
.5 .5 -.5 -.5
.5 -.5 .5 .5
.5 -.5 .5 -.5
.5 -.5 -.5 .5
.5 -.5 -.5 -.5
-.5 .5 .5 .5
-.5 .5 .5 -.5
-.5 .5 -.5 .5
-.5 -.5 .5 .5
-.5 -.5 .5 -.5
-.5 -.5 -.5 .5
-.5 -.5 -.5 -.5
-.5 0 0 0
.5 0 0 0
0 -.5 0 0
0 .5 0 0
0 0 -.5 0
0 0 .5 0
0 0 0 -.5
0 0 0 .5
userve% tersm_mm new_coeffs.in  datapoints.in datapoints.out
userve% cat datapoints.out
CS_Alt CS_Vel CS_Az CS_CC Model_1 Model_2 Model_3 Model_4 Model_5 Model_6 Model_7
1.000000 1.000000 1.000000 1.000000  553  628  613  629  635  629  650
1.000000 1.000000 1.000000 -1.000000  225  209  192  203  200  195  193
1.000000 1.000000 -1.000000 1.000000  395  341  342  341  346  339  342
1.000000 1.000000 -1.000000 -1.000000  67  51  58  56  38  56  56
1.000000 -1.000000 1.000000 1.000000  245  283  260  248  271  247  248
1.000000 -1.000000 1.000000 -1.000000  106  53  73  70  63  72  71
1.000000 -1.000000 -1.000000 1.000000  171  129  114  126  116  113  118
1.000000 -1.000000 -1.000000 -1.000000  32  28  47  36  34  49  41
-1.000000 1.000000 1.000000 1.000000  497  467  419  454  454  424  437
-1.000000 1.000000 1.000000 -1.000000  280  209  213  210  217  219  217
-1.000000 1.000000 -1.000000 1.000000  340  343  338  333  333  328  334
-1.000000 1.000000 -1.000000 -1.000000  123  114  137  128  119  134  123
-1.000000 -1.000000 1.000000 1.000000  189  171  172  184  179  172  183
-1.000000 -1.000000 1.000000 -1.000000  161  102  115  104  91  111  101
-1.000000 -1.000000 -1.000000 1.000000  116  82  95  88  92  99  97
-1.000000 -1.000000 -1.000000 -1.000000  88  42  64  67  26  64  58
0.000000 0.000000 0.000000 0.000000  224  533  570  544  550  572  555
-1.000000 0.000000 0.000000 0.000000  224  490  523  505  509  523  511
1.000000 0.000000 0.000000 0.000000  224  514  476  505  533  476  515
0.000000 -1.000000 0.000000 0.000000  138  228  191  212  247  191  218
0.000000 1.000000 0.000000 0.000000  310  412  439  423  431  439  430
0.000000 0.000000 -1.000000 0.000000  166  385  371  423  464  372  430
0.000000 0.000000 1.000000 0.000000  282  508  683  608  590  685  616
0.000000 0.000000 0.000000 -1.000000  135  430  249  301  361  251  306
0.000000 0.000000 0.000000 1.000000  313  635  633  572  566  637  584
0.500000 0.500000 0.500000 0.500000  364  620  781  667  627  625  625
```

```
0.500000 0.500000 0.500000 -0.500000  238  467  445  458  414  414  418
0.500000 0.500000 -0.500000 0.500000  296  524  567  537  530  535  523
0.500000 -0.500000 0.500000 0.500000  244  491  501  485  648  679  651
0.500000 0.500000 -0.500000 -0.500000  170  397  299  356  342  327  343
0.500000 -0.500000 0.500000 0.500000  244  491  501  485  648  679  651
0.500000 -0.500000 0.500000 -0.500000  165  386  299  346  487  476  488
0.500000 -0.500000 -0.500000 0.500000  197  422  357  394  578  583  569
0.500000 -0.500000 -0.500000 -0.500000  118  342  218  283  442  417  437
-0.500000 0.500000 0.500000 0.500000  350  580  726  624  618  581  603
-0.500000 0.500000 0.500000 -0.500000  252  461  485  465  443  450  449
-0.500000 0.500000 -0.500000 0.500000  282  512  580  530  550  546  539
-0.500000 -0.500000 0.500000 0.500000  231  457  463  457  650  635  644
-0.500000 -0.500000 0.500000 -0.500000  179  386  332  360  517  526  525
-0.500000 -0.500000 -0.500000 0.500000  183  404  349  380  597  574  582
-0.500000 -0.500000 -0.500000 -0.500000  132  346  250  305  476  474  480
-0.500000 0.000000 0.000000 0.000000  224  519  565  534  570  610  583
0.500000 0.000000 0.000000 0.000000  224  531  539  534  549  582  559
0.000000 -0.500000 0.000000 0.000000  181  434  391  415  567  584  571
0.000000 0.500000 0.000000 0.000000  267  525  593  546  506  515  507
0.000000 0.000000 -0.500000 0.000000  195  480  474  489  513  519  512
0.000000 0.000000 0.500000 0.000000  253  542  644  584  575  593  581
0.000000 0.000000 0.000000 -0.500000  180  481  412  441  448  447  450
0.000000 0.000000 0.000000 0.500000  269  584  658  590  609  613  603
```

TEST 8: Files specified in command line do not exist.

```
userve% tersm_mm coeffs.bad in.bad out.bad
ERROR: failed to open coeffs file.
```

TEST 9: Files specified by user in response to prompt do not exist.

```
userve%  tersm_mm
Enter Metamodel Coefficient file name: coeffs.bad
ERROR: failed to open coeffs file.
userve%  tersm_mm
Enter Metamodel Coefficient file name: new_coeffs.in
Enter test case data option (? lists options):F
Enter test case file name: in.bad
ERROR: cannot open test case parameter file
Enter test case data option (? lists options):Q
```

TEST 10: Number of metamodel coefficients sets in the coefficients file does not match the number of models permitted in the program.

```
userve% cat one_coeff.in
1
224.118 0.0 85.75 57.75 89.0 0.0 0.0 27.75 21.0 47.25 0.0 0.0 0.0 0.0 0.0
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
userve% tersm_mm one_coeff.in
ERROR: Number of models in file does not match actual_n_models!!
```

TEST 11: Number of test cases in test case parameter file exceeds storage allocated for test case parameters.

```
userve% cat test2big.in
1021
1.0 0.0 0.0 0.0
userve% tersm_mm new_coeffs.in test2big.in
WARNING: Number of cases in file exceeds MAX_CASES!!
Enter output file name: test.out
userve% cat test.out
CS_Alt CS_Vel CS_Az CS_CC Model_1 Model_2 Model_3 Model_4 Model_5 Model_6 Model_7
1.000000 0.000000 0.000000 0.000000  224  514  476  505  533  476  515
0.000000 0.000000 0.000000 0.000000  224  533  570  544  550  572  555
0.000000 0.000000 0.000000 0.000000  224  533  570  544  550  572  555
0.000000 0.000000 0.000000 0.000000  224  533  570  544  550  572  555
    .        .        .        .      .    .    .    .    .    .    .
    .        .        .        .      .    .    .    .    .    .    .
    .        .        .        .      .    .    .    .    .    .    .
```

TEST 12: User response to prompt is not a valid option.

```
userve% tersm_mm new_coeffs.in
Enter test case data option (? lists options):3
ERROR: Invalid input.
Enter test case data option (? lists options):H
ERROR: Invalid input.
Enter test case data option (? lists options):1
Input variable to vary (? to list options): ?
   (A)ltitude
   (V)elocity
   A(Z)imuth
   (C)hannel capacity
Input variable to vary (? to list options): Q
ERROR: Variable option is not valid.
Input variable to vary (? to list options): A
Enter increment for altitude: -.2
ERROR: increment must be in range (0,1].
Enter increment for altitude: 0.0
ERROR: increment must be in range (0,1].
Enter increment for altitude: 4
```

```
ERROR: increment must be in range (0,1].
Enter increment for altitude: .5
Input centered, scaled velocity: [-1,1]:-2
ERROR: velocity parameter out of range.
Input centered, scaled velocity: [-1,1]:2
ERROR: velocity parameter out of range.
Input centered, scaled velocity: [-1,1]:0
Input centered, scaled azimuth: [-1,1]:0
Input centered, scaled chan cap.: [-1,1]:0
Enter output file name: test.out
userve% cat test.out
CS_Alt CS_Vel CS_Az CS_CC Model_1 Model_2 Model_3 Model_4 Model_5 Model_6 Model_7
-1.000000 0.000000 0.000000 0.000000  224  490  523  505  509  523  511
-0.500000 0.000000 0.000000 0.000000  224  519  565  534  570  610  583
0.000000 0.000000 0.000000 0.000000  224  533  570  544  550  572  555
0.500000 0.000000 0.000000 0.000000  224  531  539  534  549  582  559
1.000000 0.000000 0.000000 0.000000  224  514  476  505  533  476  515

userve% !t
tersm_mm  new_coeffs.in
Enter test case data option (? lists options):2
Input two variables to vary (? to list options): QQ
ERROR: First variable option is not valid.
Input two variables to vary (? to list options): QA
ERROR: First variable option is not valid.
Input two variables to vary (? to list options): AQ
ERROR: second variable option is not valid.
Input two variables to vary (? to list options): AA
ERROR: second variable option is not valid.
Input two variables to vary (? to list options): AV
Enter increment for altitude: -.5
ERROR: increment must be in range (0,1].
Enter increment for altitude: 0.0
ERROR: increment must be in range (0,1].
Enter increment for altitude: 2
ERROR: increment must be in range (0,1].
Enter increment for altitude: .00001
ERROR: too small increment generates too many cases.
Enter increment for altitude: .1
Enter increment for velocity: -2
ERROR: increment must be in range (0,1].
Enter increment for velocity: 0.0
ERROR: increment must be in range (0,1].
Enter increment for velocity: 1.2
ERROR: increment must be in range (0,1].
Enter increment for velocity: .00001
ERROR: too small increment generates too many cases.
Enter increment for velocity: .001
ERROR: too small increment generates too many cases.
Enter increment for velocity: .1
Input centered, scaled azimuth: [-1,1]:-2
```

```
ERROR: azimuth parameter out of range.
Input centered, scaled azimuth: [-1,1]:1
Input centered, scaled chan cap.: [-1,1]:-2
ERROR: chan cap. parameter out of range.
Input centered, scaled chan cap.: [-1,1]:2
ERROR: chan cap. parameter out of range.
Input centered, scaled chan cap.: [-1,1]:1
Which metamodel should be used (1-7)? 0
ERROR: invalid metamodel number.
Which metamodel should be used (1-7)? 8
ERROR: invalid metamodel number.
Which metamodel should be used (1-7)? 7
Enter output file name: test.out
userve% cat test.out
 -1.00 -0.90 -0.80 -0.70 -0.60 -0.50 -0.40 -0.30 -0.20 -0.10 0.00 0.10 0.20 0.30 0.40 0.50 0.60
0.70 0.80 0.90 1.00
-1.00  183  286  376  448  498  530  547  552  548  541  531  521  513  506  502  499  495  490
480  464  437
-0.90  207  316  411  485  539  572  589  595  592  584  574  564  556  550  545  542  539  533
524  507  479
-0.80  224  337  435  512  567  601  619  625  622  615  605  595  587  581  576  573  570  565
555  538  509
-0.70  235  350  450  529  585  620  639  645  643  635  625  616  608  601  597  595  592  587
577  560  531
-0.60  241  358  459  538  595  631  650  657  655  647  638  629  621  615  611  608  606  601
592  574  545
-0.50  243  361  463  543  600  637  656  663  661  654  645  636  628  622  619  617  614  610
601  584  555
-0.40  243  361  464  544  602  639  658  666  664  657  648  640  632  627  623  621  620  615
607  589  561
-0.30  243  361  463  544  601  639  659  666  665  659  650  642  634  629  626  625  623  619
611  594  565
-0.20  242  360  462  543  601  639  659  667  666  660  652  643  636  632  629  628  626  623
614  598  569
-0.10  241  359  462  543  602  640  660  668  668  662  654  646  639  635  633  632  630  627
619  603  574
0.00  242  360  463  545  604  642  663  672  671  666  658  650  644  640  638  637  636  633
625  609  581
0.10  244  363  466  549  608  647  668  677  677  672  664  657  651  647  645  645  644  641
634  618  590
0.20  247  367  471  554  614  653  675  684  685  680  673  665  660  656  654  654  654  652
644  629  601
0.30  251  372  478  561  622  662  684  694  694  690  683  676  670  667  666  666  666  664
657  641  613
0.40  256  378  485  570  631  671  694  704  705  701  694  687  682  679  678  679  679  677
670  655  627
0.50  261  385  493  578  640  681  704  715  716  712  705  699  694  691  691  692  692  691
684  669  641
0.60  265  390  499  585  648  689  713  724  725  722  716  709  705  702  702  703  704  703
697  682  654
```

0.70 268 393 502 589 653 695 719 730 732 728 722 717 712 710 710 712 713 712
706 691 663
0.80 266 392 501 588 652 694 719 730 732 729 724 718 714 712 713 715 716 716
710 695 668
0.90 260 385 494 580 644 686 711 722 725 722 717 712 708 707 707 709 711 711
706 692 664
1.00 248 370 477 562 625 667 692 704 706 704 699 694 691 690 691 693 696 696
691 677 650

G.1.5 User's Notes

These users' notes provide information on the execution of the TERSM Metamodel program, providing information on how to execute and use the program, the structures of the files that are input to the program, and the structure of the output file.

There are two main inputs that the user must supply to the program: the metamodel coefficients and the test case parameters. The metamodel cofficients must be stored in a file; the test case parameters can either be provided in a file, or the program can automatically generate a series of test cases based on user-supplied specifications. Data is written to an output file, whose name must also be provided by the user.

G.1.5.1 Command Line Interface

File names can be provided in the command line used to initiate the program, or can be input by the user in response to program prompts. The format of the command line is as follows:

%tersm_mm [*filename1*] [*filename2*] [*filename3*]

where
filename1 is the name of the metamodel coefficients file;
filename2 is the name of the test data input file; and
filename3 is the name of the output file.

G.1.5.2 Program User Interface

The user interface to the TERSM Metamodel program consists of text prompts provided by the program, and text responses by the user. User response include single letter codes which designate certain options, and names of files.

If no file names are included in the command line, the following text is output by the program: "Enter Metamodel Coefficient file name:". The user must respond with the name of a file. The program then opens the file and checks the first field, which contains the number of models for which coefficients are defined. Since the TERSM metamodel study involved seven (7) metamodels, this number must be seven.

The next step of the program is to establish the test case parameters. There are several ways in which this step is accomplished. As noted in the preceding section, the file name can be included in the command line. If the file name is not included in the command line, the following

text is output by the program: "Enter test case data option (? lists options):". The user then must input a single character (followed by carriage return) to indicate which of the following options is to be executed:

?: Lists these options

C: The coefficients that were read in from the coefficients file are output to the terminal in the form of the metamodel equations

F: The test case parameters are contained in a file

1: The test case parameters are to be automatically generated; one of the input parameters is incremented by a fixed value while the other three parameters are constant, and data is generated for each of the metamodels.

2: The test case parameters are to be automatically generated; two of the input parameters are incremented by separate fixed values while the other two parameters are constant, and data is generated for one metamodel

Q: Program execution is aborted

Program response to each of these options is described in a paragraph below.

List options: The options listed above are output to the user's terminal, as follows:

```
C: print out coefficients in metamodel eqn form
F: use test cases defined in a file
1: vary a single parameter
2: vary two parameters
Q: quit, do not generate data
```

The user is then prompted for the input option.

Print coefficients: The metamodel coefficients are output to the user's terminal in the form of the metamodel equations, e.g.

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_{12} x_1 x_2 + \ldots$$

Terms with a coefficient of zero are not printed out. The primary uses for this option includes generation of the metamodel equation text, and checking to ensure that the correct coefficients are stored in the coefficients file. Program execution terminates after the equations have been output to the user's terminal.

Use test cases defined in a file: The following prompt is output to the user's terminal: "Enter test case file name:" The user must then input the name of the test case parameters file.

Generate single varying parameter test cases: This particular option is intended to be used to generate data that can be plotted to compare multiple metamodels. In this option, the program generates as output an array in which the rows are the individual test cases, and the columns are

the metamodels. In order to automatically generate the test cases, the user must input which of the four parameters is to be varied, the increment by which the parameter is increased for each test case, and the values of the remaining parameters. The following prompt is initially output to the user's terminal: "Input variable to vary (? to list options):" The user must then input one of the four parameters. If a '?' is input, the following is output to the terminal:

(A)ltitude
(V)elocity
A(Z)imuth
(C)hannel capacity

Upon a valid input, the user is prompted to enter the increment for the varying parameter with the the text "Enter increment for *parameter_name*:", where *parameter_name* is the name of the parameter to be varied. The increment must be in the range (0,1], and large enough so that the number of test cases does not overflow the available space for storing test case parameters. Next the user is prompted for the values of the remaining three parameters. Since parameters are centered, scaled values, they must all fall within the range [-1, 1]. The prompt in each case is "Input centered, scaled *parameter_name*: [-1,1]:", where *parameter_name* is the name of one of the three remaining parameters.

Generate two varying parameter test cases: This particular option is intended to be used to generate data that can be plotted to show a two-dimensional response surface of a particular metamodel. In this option, the program generates as output an array in which the rows represent the various values of one parameter, and the columns represent various values of the other parameter. In order to automatically generate the test cases, the user must input which two of the four parameters are to be varied, the increments by which the parameters are to be increased for each test case, the values of the remaining parameters, and the metamodel to be used to generate the data. The following prompt is initially output to the user's terminal: "Input two variables to vary (? to list options):" The user must then input two of the four parameters. If a '?' is input, the list variables is output as in the case of single variable input (see the preceding paragraph).

Upon a valid input, the user is prompted to enter the increments for each of the varying parameters with the the text "Enter increment for *parameter_name*:", where *parameter_name* is the name of the parameter to be varied. The increments must both be in the range (0,1], and large enough so that the number of test cases does not overflow the available space for storing test case parameters. Next the user is prompted for the values of the remaining two parameters. Since parameters are centered, scaled values, they must all fall within the range [-1, 1]. The prompt in each case is "Input centered, scaled *parameter_name*: [-1,1]:", where *parameter_name* is the name of one of the two remaining parameters. The following prompt is then output to the user's terminal: "Which metamodel should be used (1-7)?". The user must then enter a number indicating the metamodel to be used to generate the data.

Exit: Normally the user cannot gracefully escape the input of a test case data option until a valid input in entered. The 'Q' option allows the user to gracefully abort the program. If the user enters 'Q' as the option, program execution terminates.

Upon completion of the test case input processing, if the output file name has not been provided on the command line, the following prompt is output to the user's terminal: "Enter output file name:" The user must then input a valid output file name.

G.1.5.3 File Structures

There are three major files that are used by the TERSM_MM program: the coefficients file, the test case parameters files, and the output file. All three files are ASCII text files. Each of the subsections below provides a summary of one of the file formats.

G.1.5.3.1 Coefficients File

The coefficients file contains the metamodel coefficients. The first field in the file is the number of metamodels for which coefficients are defined; the remaining parameters are the coefficients. For each metamodel, there must be twenty-eight (28) values in the files (corresponding to β_0, β_1, β_2, β_3, β_4, β_{12}, β_{13}, β_{14}, β_{23}, β_{24}, β_{34}, β_{123}, β_{124}, β_{134}, β_{234}, β_{1234}, β_{11}, β_{22}, β_{33}, β_{44}, β_{111}, β_{222}, β_{333}, β_{444}, β_{1111}, β_{2222}, β_{3333}, β_{4444}). If there is no term in the metamodel equation, the value of zero must be included in the coefficients file for that term.

G.1.5.3.2 Test Case Input File

The test case parameters file contains the test cases for which the metamodels are to be run. Output for test cases stored in a parameter is generated in the same format as the single varying parameter case; that is, as an array where each row represents a test case, and each column represents a metamodel. However, there is no restriction on the values of the parameters. Different test cases could have different values for all four parameters, and/or parameter values could be incremented at varying values.

The first field in a test case parameters file is the number of test cases in the file. If this number exceeds the amount of storage available for storing the test case parameters, only the number of test cases that can be stored are read in and executed. The remaining values in the test case parameter file are parameter values for each test case, beginning with altitude, followed by velocity, azimuth, and channel capacity. Parameter values must be in the range [-1, 1].

G.1.5.3.3 Output File

The output file, also referred to as the results file, is an ASCII text file. There are two formats in which the data is output, depending on which option is selected by the user.

The first output format is generated when the test case parameters are read in from a file, or when the user selects a single varying parameter for automatically generated test cases. In this format, the output consists of an array with columns as follows:

Centered, scaled altitude value

Centered, scaled velocity value

Centered, scaled azimuth value

Centered, scaled channel capacity value

Metamodel output 1

-
-
-

The first row in the output contains column headers; each remaining row represents an individual test case.

For the output format with two varying parameters, the output array contains data generated by a single metamodel. The rows represent various values for the first varying parameter, with the columns representing values of the second parameter. For example, suppose the user had specified "AV" as the variables to vary, and had input an increment of 0.25 for the altitude increment, and .5 for the velocity increment. The output would be as follows:

	-1.00	-0.50	0.0	0.5	1.0
-1.00	metamodel value for Alt = -1, Vel = -1	metamodel value for Alt = -1, Vel = -0.5			
-0.75	metamodel value for Alt = -0.75, Vel = -1				
-0.50					
-0.25					
0.0					
0.25					
0.50					
0.75					
1.00					

The output is designed to be read into Microsoft Excel™ (or other compatable spreadsheet programs) to allow the generation of graphs of the data. From Excel™, open the output file. The first dialog box will include options for how the text is organized; choose the Delimited option.

The second dialog box includes options for how the text is delimited; choose the Spaces option. The third and final dialog box includes options for data format; choose the General option. The data is now in Excel™ spreadsheet form. From this point, graphs can be created from the data using the Excel™ Chart option.

G.1.5.4 Error Messages

ERROR: output option of *option* not valid.

The user input option is not a valid option. Input one of the valid options. Input "?" to see valid options for this prompt.

ERROR: failed to open coeffs file.

The coefficients file specified by the user could not be opened. Check to ensure that the file exists in the specified directory path.

ERROR: number of models in file does not match number expected (*n*).

The number of models for which coefficients are defined in the coefficients file (as specified in the first field of the file) does not match the number of expected models. The expected number is in paratheses at the end of the message. The program is currently set for seven (7) sets of coefficients, based on the seven metamodels defined in Zeimer et. al. (1993). Check to make sure that the correct number of model coefficients is defined in the file. This error also often appears in cases where the specified file is not a coefficients file; check the input file name to ensure that it is the intended coefficients file.

ERROR: invalid test case option.

The test case option input by the user is not a valid option. Input "?" to see valid responses for this prompt, or re-input the variable to be varied.

ERROR: failed to open test case parameter file

The test case parameter file could not be opened. Check to ensure that the file exists in the specified directory path.

WARNING: number of cases in file exceeds available storage (*n*).

The number of test cases exceeds the storage in the program allocated for test case parameters. The maximum number is included in paraentheses at the end of the message. Program execution continues using as many test cases as possible.

ERROR: variable option is not valid.

The user input variable designator is not valid. Input one of the valid variable designators. Input "?" to see valid responses for this prompt, or re-input the variable to be varied.

ERROR: increment must be in range (0,1].

The increment value for the varying parameter must be greater than 0.0 and less than or equal to 1.0. Re-input a valid increment value.

ERROR: too small increment generates too many cases.

One test case is generated for each increment value in the range of -1 to 1. The number of test cases generated by this increment exceeds the storage in the program allocated for test case parameters. Re-input a larger increment value.

ERROR: *parameter_name* parameter out of range.

The value input for the parameter *parameter_name* is less than 1.0 or greater than 1.0. The parameter values input to the metamodels are scaled and centered, and thus must be in the range [-1, 1]. Re-input the parameter value.

ERROR: first variable option is not valid.

The user input variable designator is not valid. Input one of the valid variable designators. Input "?" to see valid responses for this prompt, or re-input the variable to be varied.

ERROR: second variable option is not valid.

The user input variable designator for the second varying parameter is not valid. Input one of the valid variable designators. This error will also be output if the same variable is input for both varying parameters. Input "?" to see valid responses for this prompt, or re-input the variable to be varied.

ERROR: invalid metamodel number.

The metamodel number input by the user does not correspond to a valid metamodel. Re-input the number.

ERROR: could not open results file.

The output file could not be opened. Make sure that the user has write privileges to the directory, and that the specified file name is a legal file name.

G.1.6 Programmer's Notes

The high level program structure is documented in the Design Notes section above, and details of the code are documented in the in-line documentation of the source code which follows. The following information is provided as supplemental information for program maintenance and updates.

Array dimension values: There are two parameters which specify sizes for arrays in the program:

N_MODELS: the number of metamodels (plus one).

MAX_CASES: the number of test case parameters.

N_MODELS is actually one more than the number of metamodels expected to be used. For clarity of code, the arrays of metamodel coefficients are indexed by model number (e.g., for seven metamodels, the indexes 1 through 7 are used). Thus the memory locations specified by the 0th index into the coefficient arrays (e.g. beta_12[0]) are not used.

Output formats: The output format does not allow much flexibility; in fact, I considered allowing test cases with two varying parameters to be specified in a file as individual test cases (and output in the second output format). However, such a file would need to be checked to ensure that only two values are varied, and that all combinations of the varying values were entered. The effort required to ensure proper values for generating the second output format was not considered worth the benefit of that additional capability.

Make command: The makefile command for compilation of this code is as follows:

```
cc -o tersm_mm tersm_mm.c -lm
```

G.1.7 Program Listing

```
/*    TERSM Metamodel Calculation Program


      Written by:  Fred Frantz
                   Computer Sciences Corporation
                   Syracuse, NY


      Under contract F30602-94-C-0272
                     Model Abstraction Techniques


      Initial implementation:  3 October 1995

      The purpose of this program is to implement the metamodels of the
      in the paper written by Zeimer, Tew, Sargent, and Sisti, entitled
      "Metamodel Procedures for Air Engagement Simulation Models", January
      1993.  This paper describes  an approach for developing metamodels
      (mathematical approximations) of the TERSM discrete event simulation
      model.  Under the Model Abstraction Techniques contract, we are
      looking at metamodeling as an abstraction technique.  In order to
      better understand the metamodels described in Zeimer et. al., it is
      useful to generate some data using the metamodel.

      Initially, the metamodeling results will be written to a text file to
      be ported to a PC running Excel for charting; eventually, something
      like .gnuplot could be used.

      There are four inputs to the metamodels; altitude, velocity, azimuth,
```

```
     and channel capacity, all centered and scaled to be in the range
     [ -1, 1] .  The metamodels all involve summing terms which are the
     product of these parameters and a set of coefficients.

     The inputs to this program are the metamodel coefficients and the
     test case parameters.  The output is a set of metamodel values.
     The user can specify three file names in the command line as follows:
           tersm_mmf  coeff_file test_inputs test_outputs
     Alternatively, the program will prompt the user for file names not
     included in the command line.  The program also provides an option
     for the test cases to be generated automatically rather than entered
     into a file.  Two generation options are provided.  In the first
     case, one of the four inputs is varied at a fixed, user-specified
     increment, and the other three parameters are held constant at user-
     specified values.  All metamodels are executed for each test case,
     each row of the output is a test case, and each column represents
     a different metamodel.  The second autogeneration option allows the
     user to specify increments for two of the four parameters, with
     constant values for the other two parameters.  The output is an array
     with rows representing single values of the first varying paramater
     and columns representing the values of the second varying parameter.
     In this case, only one metamodel is used to generate output per run.
     (Because this output assumes fixed increments for the parameters, it
     is only utilized for autogenerated test cases; user input files are
     assumed to be in the first format.)

     There are a number of error conditions which are also checked
     throughout the code to ensure user entries are within range.
                                                                        */

#include <stdio.h>
#include <math.h>

#define N_MODELS 8   /* number of metamodels for which coefficients are
         defined (actually one more than the number of models; for code
            readability, we use indexes 1 through 7, but dimension 0..7 */
#define MAX_CASES 500                    /* maximum number of test cases */
#define FILE_NAME_LENGTH 80               /* length of file name strings */

float  beta_0[ N_MODELS] , beta_1[ N_MODELS] , beta_2[ N_MODELS] ,
       beta_3[ N_MODELS] , beta_4[ N_MODELS] , beta_12[ N_MODELS] ,
       beta_13[ N_MODELS] , beta_14[ N_MODELS] , beta_23[ N_MODELS] ,
       beta_24[ N_MODELS] , beta_34[ N_MODELS] , beta_123[ N_MODELS] ,
       beta_124[ N_MODELS] , beta_134[ N_MODELS] , beta_234[ N_MODELS] ,
       beta_1234[ N_MODELS] , beta_11[ N_MODELS] , beta_22[ N_MODELS] ,
       beta_33[ N_MODELS] , beta_44[ N_MODELS] , beta_111[ N_MODELS] ,
       beta_222[ N_MODELS] , beta_333[ N_MODELS] , beta_444[ N_MODELS] ,
       beta_1111[ N_MODELS] , beta_2222[ N_MODELS] , beta_3333[ N_MODELS] ,
       beta_4444[ N_MODELS] ;                    /*  metamodel coefficients */
float  y;                                            /* metamodel output */
int    yint;                                         /* for final output */
float  x[ 4] ;                                /* metamodel input parameters */
float  cs_altitude[ MAX_CASES] ,
       cs_velocity[ MAX_CASES] ,
       cs_azimuth[ MAX_CASES] ,
```

```
        cs_channel_capacity[ MAX_CASES] ;          /* test case parameters */
float   increment, increment_2;                     /* increments for auto
                                                      generated test cases */
int     actual_n_models;                  /* actual number of metamodels */
int     f_models;                          /* number of models for which
                                                    coefficients are in file */
int     model_num;                /* metamodel number for generating data */
int     i_case, n_cases;      /* index and number of test cases to be run */
int     n_v1_cases, n_v2_cases;      /* number of test cases for varying
                                        parameters 1 AND 2, respectively */
int     j1, j2;                                              /*loop indexes */
int     output_option;                    /* output option: 1= 7 columns
                                  per test case one for each metamodel
                               2= each row varying param 1, each column
                                    varying param 2, one metamodel;
                                  also used to flag open file failures */
int     varying_param, varying_param_2;
                                          /* indicates which parameters
                                                        are being varied */
FILE    *coefficient_file;           /* file with metamodel coefficients */
FILE    *case_file;                    /* file with test case parameters */
FILE    *results_file;              /* output file with metamodel results */
char    case_file_name[ FILE_NAME_LENGTH] ;
                              /* name of file with test case parameters */
char    results_file_name[ FILE_NAME_LENGTH] ;      /* name of output file */

main(argc, argv)
int   argc;
char  *argv[ ] ;
 {
 /*
      Initialize coefficients.
      Initial test cases.
      Open results file.
                                                                        */
   actual_n_models = N_MODELS -1;
   init_coeffs(argc, argv[ 1] ) ;
   if (output_option > 0) {
     init_test_case(argc, argv[ 2] ) ;
    }
   /*
        IF there wasn't a problem getting the input data
             THEN open the output file.
                                                                        */
   if (output_option > 0) {
     init_results_file(argc, argv[ 3] ) ;
     if (output_option == 1) {
     /*
        Calculate the metamodel output for each of the 7 models for
        each of the test cases.

        First print the column headers in the output file.
                                                                        */
       fprintf (results_file, "CS_Alt CS_Vel CS_Az CS_CC ");
       for (model_num=1;model_num<=actual_n_models;model_num++)   {
```

```
  fprintf (results_file, "Model_%d ", model_num);
 }
fprintf (results_file, "\n");
/*
     Generate the metamodel outputs.
                                                                    */
for (i_case=0;i_case<=n_cases-1;i_case++) {
  x[ 0]  = cs_altitude[ i_case] ;
  x[ 1]  = cs_velocity[ i_case] ;
  x[ 2]  = cs_azimuth[ i_case] ;
  x[ 3]  = cs_channel_capacity[ i_case] ;
  fprintf (results_file,"%f %f %f %f ", x[ 0] , x[ 1] , x[ 2] , x[ 3] );
  for (model_num=1;model_num<=actual_n_models;model_num++)   {
     y = beta_0[ model_num]  +
         beta_1[ model_num] * x[ 0]  +
         beta_2[ model_num] * x[ 1]  +
         beta_3[ model_num] * x[ 2]  +
         beta_4[ model_num] * x[ 3]  +
         beta_12[ model_num] * x[ 0] * x[ 1]  +
         beta_13[ model_num] * x[ 0] * x[ 2]  +
         beta_14[ model_num] * x[ 0] * x[ 3]  +
         beta_23[ model_num] * x[ 1] * x[ 2]  +
         beta_24[ model_num] * x[ 1] * x[ 3]  +
         beta_34[ model_num] * x[ 2] * x[ 3]  +
         beta_123[ model_num] * x[ 0] * x[ 1] * x[ 2]  +
         beta_124[ model_num] * x[ 0] * x[ 1] * x[ 3]  +
         beta_134[ model_num] * x[ 0] * x[ 2] * x[ 3]  +
         beta_234[ model_num] * x[ 1] * x[ 2] * x[ 3]  +
         beta_1234[ model_num] * x[ 0] * x[ 1] * x[ 2] * x[ 3]  +
         beta_11[ model_num] * x[ 0] * x[ 0]  +
         beta_22[ model_num] * x[ 1] * x[ 1]  +
         beta_33[ model_num] * x[ 2] * x[ 2]  +
         beta_44[ model_num] * x[ 3] * x[ 3]  +
         beta_111[ model_num] * x[ 0] * x[ 0] * x[ 0]  +
         beta_222[ model_num] * x[ 1] * x[ 1] * x[ 1]  +
         beta_333[ model_num] * x[ 2] * x[ 2] * x[ 2]  +
         beta_444[ model_num] * x[ 3] * x[ 3] * x[ 3]  +
         beta_1111[ model_num] * x[ 0] * x[ 0] * x[ 0] * x[ 0]  +
         beta_2222[ model_num] * x[ 1] * x[ 1] * x[ 1] * x[ 1]  +
         beta_3333[ model_num] * x[ 2] * x[ 2] * x[ 2] * x[ 2]  +
         beta_4444[ model_num] * x[ 3] * x[ 3] * x[ 3] * x[ 3] ;
    /*
           Now adjust the metamodel output for models based on
           the square root or ln of y (see Zeimer et. al.).
           Also convert to integer (since we are modeling a
           discrete number of emitters.
                                                                    */
    if (model_num == 3 || model_num == 6)  {
       y = exp(y);
       yint = y + 0.5;
     }
    else if (model_num == 4 || model_num == 7)  {
       y = y * y ;
       yint = y + 0.5;
     }
```

```
        else {
          yint = y + 0.5;
        }
        /*
             Output metamodel calculation to the results file.
                                                                    */
        fprintf(results_file, " %d ", yint);

      } /* end model_num loop */

     fprintf (results_file, "\n");

   }                                  /* end individual test case loop */
 }                                              /* end output option 1 */
else if (output_option == 2) {
  /*
     First print the column headers in the output file.  Each
     column header is a value of the second varying parameter.
                                                                    */
   fprintf (results_file, " ");
   x[ varying_param_2] = -1.0;
   for (j2=0;j2<n_v2_cases;j2++) {
     fprintf (results_file, "%.2f ", x[ varying_param_2] );
     x[ varying_param_2] += increment_2;
    }
   i_case = 0;
   while (x[ varying_param_2] <= 1.0);
   fprintf (results_file, "\n");
  /*
       Generate the metamodel outputs.
                                                                    */
   for (j1=0;j1<n_v1_cases;j1++) {
     for (j2=0;j2<n_v2_cases;j2++) {
       x[ 0] = cs_altitude[ i_case] ;
       x[ 1] = cs_velocity[ i_case] ;
       x[ 2] = cs_azimuth[ i_case] ;
       x[ 3] = cs_channel_capacity[ i_case] ;
       if (j2 == 0 ) {     /* we are at the beginning of a new line */
         fprintf (results_file, "%.2f ", x[ varying_param] );
        }
       y = beta_0[ model_num] +
           beta_1[ model_num] * x[ 0] +
           beta_2[ model_num] * x[ 1] +
           beta_3[ model_num] * x[ 2] +
           beta_4[ model_num] * x[ 3] +
           beta_12[ model_num] * x[ 0] * x[ 1] +
           beta_13[ model_num] * x[ 0] * x[ 2] +
           beta_14[ model_num] * x[ 0] * x[ 3] +
           beta_23[ model_num] * x[ 1] * x[ 2] +
           beta_24[ model_num] * x[ 1] * x[ 3] +
           beta_34[ model_num] * x[ 2] * x[ 3] +
           beta_123[ model_num] * x[ 0] * x[ 1] * x[ 2] +
           beta_124[ model_num] * x[ 0] * x[ 1] * x[ 3] +
           beta_134[ model_num] * x[ 0] * x[ 2] * x[ 3] +
           beta_234[ model_num] * x[ 1] * x[ 2] * x[ 3] +
```

```
                beta_1234[ model_num] * x[ 0] * x[ 1] * x[ 2] * x[ 3]  +
                beta_11[ model_num] * x[ 0] * x[ 0]  +
                beta_22[ model_num] * x[ 1] * x[ 1]  +
                beta_33[ model_num] * x[ 2] * x[ 2]  +
                beta_44[ model_num] * x[ 3] * x[ 3]  +
                beta_111[ model_num] * x[ 0] * x[ 0] * x[ 0]  +
                beta_222[ model_num] * x[ 1] * x[ 1] * x[ 1]  +
                beta_333[ model_num] * x[ 2] * x[ 2] * x[ 2]  +
                beta_444[ model_num] * x[ 3] * x[ 3] * x[ 3]  +
                beta_1111[ model_num] * x[ 0] * x[ 0] * x[ 0] * x[ 0]  +
                beta_2222[ model_num] * x[ 1] * x[ 1] * x[ 1] * x[ 1]  +
                beta_3333[ model_num] * x[ 2] * x[ 2] * x[ 2] * x[ 2]  +
                beta_4444[ model_num] * x[ 3] * x[ 3] * x[ 3] * x[ 3] ;
            /*
                Now adjust the metamodel output for models based on the
                square root or ln of y (see Zeimer et. al.) and convert
                to integer because we are modeling a discrete number of
                emitters.
                                                                        */
             if (model_num == 3 || model_num == 6)  {
               y = exp(y);
               yint = y + 0.5;
              }
             else if (model_num == 4 || model_num == 7)  {
               y = y * y ;
               yint = y + 0.5;
              }
             else {
               yint = y + 0.5;
              }
              /*
                   Output metamodel calculation to the results file.
                                                                        */
             fprintf(results_file, " %d ", yint);
             i_case++;
          }                                        /* end varying param 2 loop */
         fprintf (results_file, "\n");
         x[ varying_param_2] = -1.0;
       }
     }                                              /* end output option 2 */
    /*
         Should not get here, but just in case.
                                                                        */
    else {
      printf ("ERROR: output option of %d not valid.\n", output_option);
     }
   }                                             /* end of if output_option > 0 */
}  /* end main */


/*
      INIT_COEFFS reads in the metamodel coefficients from a file.
        nclp  is the number of command line parameters (argc in main)
        fname is the first command line parameter, which should be the
```

```
            name of the metamodel coefficient file.
                                                                        */

init_coeffs(int nclp, char fname[ FILE_NAME_LENGTH] )
 {
  char   coeff_file_name[ FILE_NAME_LENGTH] ;
                            /* name of file with metamodel coefficients */

  /*
       IF the file name is not in the command line
            THEN copy the name from the command line buffer
            ELSE prompt the user for the name.
                                                                        */
  if (nclp > 1) {
    strcpy(coeff_file_name, fname);
   }
  else {
    printf ("Enter Metamodel Coefficient file name: ");
    scanf ("%s", &coeff_file_name);
   }
  /*
       Open the file
       IF not opened
            THEN print error messages
                                                                        */
  if ((coefficient_file=fopen(coeff_file_name, "r")) == NULL) {
    printf  ("ERROR: failed to open coeffs file.\n");
    output_option = 0;
    }
  /*
             ELSE read in coefficients from the file
                                                                        */
  else {
    fscanf(coefficient_file,"%d", &f_models);
    if (f_models == actual_n_models && f_models > 0)   {
      for (model_num=1;model_num <= f_models;model_num++)  {
         fscanf(coefficient_file,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f",
           &beta_0[ model_num] ,
           &beta_1[ model_num] , &beta_2[ model_num] , &beta_3[ model_num] ,
           &beta_4[ model_num] , &beta_12[ model_num] , &beta_13[ model_num] ,
           &beta_14[ model_num] , &beta_23[ model_num] , &beta_24[ model_num] ,
           &beta_34[ model_num] , &beta_123[ model_num] , &beta_124[ model_num] ,
           &beta_134[ model_num] , &beta_234[ model_num] );
         fscanf(coefficient_file,"%f %f %f %f %f %f %f %f %f %f %f %f %f",
           &beta_1234[ model_num] ,
           &beta_11[ model_num] , &beta_22[ model_num] , &beta_33[ model_num] ,
           &beta_44[ model_num] , &beta_111[ model_num] , &beta_222[ model_num] ,
           &beta_333[ model_num] , &beta_444[ model_num] , &beta_1111[ model_num] ,
           &beta_2222[ model_num] , &beta_3333[ model_num] ,
           &beta_4444[ model_num] );
       }
    output_option = 1;
     }
    else  {
```

```
      printf ("ERROR: number of models in file does not match number expected
(%d).\n", N_MODELS-1);
      output_option = 0;
    }
  }
}   /* end INIT_COEFFS  */



/*
     INIT_TEST_CASE reads in the parameters of the test case from
     a file.
       nclp  is the number of command line parameters (argc in main)
       fname is the second command line parameter, which should be the
             name of the test case file.
                                                                        */

init_test_case(int nclp, char fname[ FILE_NAME_LENGTH] )
{
  char   test_case_option[ 80] ;
           /* indicates which test case data generation option to use */
  char   var_option[ 80] ;           /* indicates which variable to vary */
        /* Even though I only care about the first character, I use an
           80 character buffer and scanf to make sure the buffer is
           flushed each time the user must input an option.             */
  int    valid_input;    /* indicates whether user entered valid input */
  char   prompt_string[ 4][ 10] ;  /* strings describing variable options */
  int    p;                /* index of which parameter being processed */
  float  intermediate;       /* used to calculate number of test cases */

  /*
       IF the file name is in the command line
            THEN copy the name from the command line buffer and
                 read data
                                                                        */

  if (nclp > 2) {
    strcpy(case_file_name, fname);
    if ((case_file=fopen(case_file_name, "r")) == NULL) {
      printf  ("ERROR: failed to open test case parameter file.\n");
      }
      else {
        fscanf (case_file,"%d", &n_cases);
        /*
             Make sure number of cases is within range.
                                                                        */
        if (n_cases > MAX_CASES)  {
          printf ("WARNING: number of cases in file exceeds available storage
(%d).\n", MAX_CASES);
          n_cases = MAX_CASES;
          }
        output_option = 1;
        for (i_case=0;i_case <= n_cases-1;i_case++)  {
           fscanf (case_file,"%f %f %f %f", &cs_altitude[ i_case] ,
              &cs_velocity[ i_case] ,
              &cs_azimuth[ i_case] ,
              &cs_channel_capacity[ i_case] );
```

```
        }
      }
    }
     /*
            ELSE prompt the user for data generation option
                                                                        */
   else {
     /*
            Keep going until a valid option is input.
                                                                        */
     test_case_option[ 0] = ' ';      /* so loop is executed at least once */
     while  (test_case_option[ 0] != 'F' && test_case_option[ 0] != '1' &&
             test_case_option[ 0] != '2' && test_case_option[ 0] != 'Q' &&
             test_case_option[ 0] != 'C') {
       printf ("Enter test case data option (? lists options):");
       scanf ("%s", &test_case_option);
       /*
              IF user wants list of options, print them out.
                                                                        */
       if (test_case_option[ 0] == '?') {
          printf ("C: print out coefficients in metamodel eqn form\n");
          printf ("F: use test cases defined in a file\n");
          printf ("1: vary a single parameter\n");
          printf ("2: vary two parameters\n");
          printf ("Q: quit, do not generate data\n");
        }                                       /* end test case option = ? */
       /*
              CASE to use file with name supplied by user.
              IF user to use a file, prompt for name, then open and
              read data.
                                                                        */
      /*
              CASE to just print out the coefficients in the meta-
              model form so we can make sure they are correct.
                                                                        */
       else if (test_case_option[ 0] == 'C') {
         print_coeffs();
         output_option = 0;
        }                               /* end else if test case option = 'C' */
       else if (test_case_option[ 0] == 'F') {
         printf ("Enter test case file name: ");
         scanf ("%s", &case_file_name);
         if ((case_file=fopen(case_file_name, "r")) == NULL) {
           printf  ("ERROR: failed to open test case parameter file.\n");
           test_case_option[ 0] = ' ';              /* forces another loop */
           }
           else {
             fscanf (case_file,"%d", &n_cases);
             /*
                 Make sure number of cases is within range.
                                                                        */
             if (n_cases > MAX_CASES)  {
             printf ("WARNING: number of cases in file exceeds available
storage (%d).\n", MAX_CASES);
               n_cases = MAX_CASES;
```

```
        }
       output_option = 1;
       for (i_case=0;i_case <= n_cases-1;i_case++)  {
          fscanf (case_file,"%f %f %f %f", &cs_altitude[ i_case] ,
             &cs_velocity[ i_case] ,
             &cs_azimuth[ i_case] ,
             &cs_channel_capacity[ i_case] );
        }
     }              /* end ELSE read in data if file can be opened */
}                          /* end else if test case option = 'F' */

/*
     CASE to automatically generate test cases with a single
     varying parameter.  Prompt user for parameter to vary and
     increment.
                                                                    */
else if (test_case_option[ 0] == '1') {
  /*
       First set up variable prompt strings to be used.
                                                                    */
  strcpy (prompt_string[ 0] , "altitude");
  strcpy (prompt_string[ 1] , "velocity");
  strcpy (prompt_string[ 2] , "azimuth");
  strcpy (prompt_string[ 3] , "chan cap.");
  var_option[ 0] = ' ';         /* get through loop at least once */
  while (var_option[ 0] != 'A' && var_option[ 0] != 'V' &&
         var_option[ 0] != 'Z' && var_option[ 0] != 'C') {
    printf ("Input variable to vary (? to list options): ");
    scanf ("%s", &var_option);
    /*
         If option is '?', print out the options.
                                                                    */
    if (var_option[ 0] == '?') {
       printf ("    (A)ltitude\n");
       printf ("    (V)elocity\n");
       printf ("    A(Z)imuth\n");
       printf ("    (C)hannel capacity\n");
     }                                        /* if var option is ? */
    /*
         ELSE assign varying parameter based on in[ ut.
                                                                    */
    else if (var_option[ 0] == 'A') {
      varying_param = 0;
     }
    else if (var_option[ 0] == 'V') {
      varying_param = 1;
     }
     else if (var_option[ 0] == 'Z') {
      varying_param = 2;
     }
    else if (var_option[ 0] == 'C') {
      varying_param = 3;
     }
    /*
         Unless the input is not valid.
```

```
                                                                    */
          else {
            printf ("ERROR: variable option is not valid.\n");
           }
         }          /* end while looking for valid one var option input */
        /*
             Prompt user for increment of varying parameter.
                                                                    */
        valid_input = 0;
        do {
          printf ("Enter increment for %s: ", prompt_string[ varying_param] );
          scanf (" %f", &increment);
          if (increment <= 0.0 || increment > 1.0) {
            printf ("ERROR: increment must be in range (0,1] .\n");
            }
           else {
             intermediate = 2.0/increment;
             n_cases = intermediate + 1;
             if (n_cases > MAX_CASES) {
               printf ("ERROR: too small increment generates too many
cases.\n");
              }
             else {
               valid_input = 1;
              }
            }
          }                     /* end loop looking for valid increment */
          while (!valid_input);
          /*
                Now get the constant values for the other three
                parameters that are not varying.  We will skip over
                the parameter being incremented.  In each case, make
                sure the input is the range [ -1,1] .
                                                                    */
          for (p=0;p<=3;p++) {
            if (varying_param != p) {
              valid_input = 0;
              do {
                printf ("Input centered, scaled %s: [ -1,1] :",
prompt_string[ p] );
                scanf ("%f", &x[ p] );
                if (x[ p] < -1.0 || x[ p] > 1.0) {
                  printf ("ERROR: %s parameter out of range.\n",
prompt_string[ p] );
                 }
                else {
                  valid_input = 1;
                 }
               }
              while (!valid_input);
             }
           }                    /* end loop for each of four parameters */
           /*
                The input is all collected, so generate the test
                case parameters.
```

```
                                                                    */
      i_case = 0;
      x[ varying_param] = -1.0;
      for (i_case=0;i_case<n_cases;i_case++) {
        cs_altitude[ i_case] = x[ 0] ;
        cs_velocity[ i_case] = x[ 1] ;
        cs_azimuth[ i_case] = x[ 2] ;
        cs_channel_capacity[ i_case] = x[ 3] ;
        x[ varying_param] += increment;
       }
      output_option = 1;
}                          /* end else if test case option = '1' */

/*
      CASE where test cases automatically generated by varying
      two parameters.  This case is useful in generating a
      response surface for a model.
                                                                    */
else if (test_case_option[ 0] == '2') {
  /*
       First set up variable prompt strings to be used.
                                                                    */
  strcpy (prompt_string[ 0], "altitude");
  strcpy (prompt_string[ 1], "velocity");
  strcpy (prompt_string[ 2], "azimuth");
  strcpy (prompt_string[ 3], "chan cap.");
  valid_input = 0;
  do {
    printf ("Input two variables to vary (? to list options): ");
    scanf ("%s", &var_option);
    /*
         If option is '?', print out the options.
                                                                    */
    if (var_option[ 0] == '?') {
       printf ("    (A)ltitude\n");
       printf ("    (V)elocity\n");
       printf ("    A(Z)imuth\n");
       printf ("    (C)hannel capacity\n");
     }                                  /* if one var option is ? */
    /*
         ELSE assign varying parameter based on input.
                                                                    */
    else if (var_option[ 0] == 'A') {
      varying_param = 0;
      if (var_option[ 1] == 'V') {
        varying_param_2 = 1;
        valid_input = 1;
       }
      else if (var_option[ 1] == 'Z') {
        varying_param_2 = 2;
        valid_input = 1;
       }
      else if (var_option[ 1] == 'C') {
        varying_param_2 = 3;
        valid_input = 1;
```

```
    }
    else {
      printf ("ERROR: second variable option is not valid.\n");
    }
  }
 else if (var_option[ 0] == 'V') {
   varying_param = 1;
   if (var_option[ 1] == 'A') {
     varying_param_2 = 0;
     valid_input = 1;
    }
   else if (var_option[ 1] == 'Z') {
     varying_param_2 = 2;
     valid_input = 1;
    }
   else if (var_option[ 1] == 'C') {
     varying_param_2 = 3;
     valid_input = 1;
    }
   else {
     printf ("ERROR: second variable option is not valid.\n");
    }
 }
 else if (var_option[ 0] == 'Z') {
  varying_param = 2;
  if (var_option[ 1] == 'A') {
    varying_param_2 = 0;
    valid_input = 1;
   }
  else if (var_option[ 1] == 'V') {
    varying_param_2 = 1;
    valid_input = 1;
   }
  else if (var_option[ 1] == 'C') {
    varying_param_2 = 3;
    valid_input = 1;
   }
  else {
    printf ("ERROR: second variable option is not valid.\n");
   }
 }
else if (var_option[ 0] == 'C') {
  varying_param = 3;
  if (var_option[ 1] == 'A') {
    varying_param_2 = 0;
    valid_input = 1;
   }
  else if (var_option[ 1] == 'V') {
    varying_param_2 = 1;
    valid_input = 1;
   }
  else if (var_option[ 1] == 'Z') {
    varying_param_2 = 2;
    valid_input = 1;
   }
```

```
            else {
              printf ("ERROR: second variable option is not valid.\n");
             }
           }
           /*
                Unless the input is not valid.
                                                                        */
           else {
             printf ("ERROR: first variable option is not valid.\n");
            }
         }          /* end loop looking for valid one var option input */
        while (!valid_input);
        /*
             Prompt user for increment of varying parameter.
                                                                        */
        valid_input = 0;
        do {
          printf ("Enter increment for %s: ", prompt_string[ varying_param] );
          scanf (" %f", &increment);
          if (increment <= 0.0 || increment > 1.0) {
            printf ("ERROR: increment must be in range (0,1] .\n");
            }
          else {
            intermediate = 2.0/increment;
            n_v1_cases = intermediate + 1;
            if (n_v1_cases > MAX_CASES) {
              printf ("ERROR: too small increment generates too many
cases.\n");
             }
            else {
              valid_input = 1;
             }
           }
         }                      /* end loop looking for valid increment */
        while (!valid_input);
        valid_input = 0;
        do {
          printf ("Enter increment for %s: ", prompt_string[ varying_param_2] );
          scanf (" %f", &increment_2);
          if (increment_2 <= 0.0 || increment_2 > 1.0) {
            printf ("ERROR: increment must be in range (0,1] .\n");
            }
          else {
            intermediate = 2.0/increment_2;
            n_v2_cases = intermediate + 1;
            n_cases = n_v1_cases * n_v2_cases;
            if (n_cases > MAX_CASES) {
              printf ("ERROR: too small increment generates too many
cases.\n");
             }
            else {
              valid_input = 1;
             }
            }
          }                     /* end loop looking for valid increment */
```

```
        while (!valid_input);
        /*
              Now get the constant values for the other two
              parameters that are not varying.  We will skip over
              the parameters being incremented.  In each case, make
              sure the input is the range [ -1,1] .
                                                                    */
        for (p=0;p<=3;p++) {
          if (varying_param != p && varying_param_2 != p) {
            valid_input = 0;
            do {
              printf ("Input centered, scaled %s: [ -1,1] :",
prompt_string[ p] );
              scanf ("%f", &x[ p] );
              if (x[ p] < -1.0 || x[ p] > 1.0) {
                printf ("ERROR: %s parameter out of range.\n",
prompt_string[ p] );
               }
              else {
                valid_input = 1;
               }
             }
            while (!valid_input);
          }
        }                    /* end loop for each of four parameters */
        /*
             The input is all collected, so generate the test
             case parameters.
                                                                    */
        i_case = 0;
        x[ varying_param] = -1.0;
        for (j1=0;j1<n_v1_cases;j1++) {
          x[ varying_param_2] = -1.0;
          for (j2=0;j2<n_v2_cases;j2++) {
            cs_altitude[ i_case] = x[ 0] ;
            cs_velocity[ i_case] = x[ 1] ;
            cs_azimuth[ i_case] = x[ 2] ;
            cs_channel_capacity[ i_case] = x[ 3] ;
            x[ varying_param_2] += increment_2;
            i_case++;
           }
          x[ varying_param] += increment;
         }
        /*
              Set the output option, and get the model number to use
              in the calculation, since only one is run at a time
              under this option.
                                                                    */
        output_option = 2;
        valid_input = 0;
        do {
          printf ("Which metamodel should be used (1-7)? ");
          scanf ("%d", &model_num);
          if (model_num < 1 || model_num > 7) {
            printf ("ERROR: invalid metamodel number.\n");
```

```
            }
            else {
               valid_input = 1;
            }
          }
          while (!valid_input);
      }                          /* end else if test case option = '2' */

      /*
           CASE for user to exit.
                                                                        */
      else if (test_case_option[ 0] == 'Q') {
        output_option = 0;
       }
      else {
        printf ("ERROR: invalid test case option.\n");
       }
     }                        /* end while looking for valid option input */
   }                  /* end ELSE prompt user for data generation option */
}  /* end INIT_TEST_CASE  */


/*
     INIT_RESULTS_FILE opens the output file.
        nclp  is the number of command line parameters (argc in main)
        fname is the third command line parameter, which should be the
              name of the results file.
                                                                        */
init_results_file(int nclp, char fname[ FILE_NAME_LENGTH] )
{
  /*
       IF the file name is not in the command line
            THEN copy the name from the command line buffer
            ELSE prompt the user for the name
                                                                        */
  if (nclp > 3) {
    strcpy(results_file_name, fname);
   }
  else {
    printf ("Enter output file name: ");
    scanf ("%s", &results_file_name);
   }
 if ((results_file=fopen(results_file_name, "w")) == NULL) {
    printf  ("ERROR: could not open results file.\n");
      }
}  /* end INIT_RESULTS_FILE */


/*
     PRINT_COEFFS prints out the metamodel coefficients in a meta-
     model equation format.
                                                                        */

print_coeffs ()
{
```

```
int   i;                                          /* model number index */
int   line_has_printing;  /* flag indicates whether anything printed
       on this line; keeps program from generating extra blank lines */

/*
     FOR each model
                                                                        */
for (i=1;i<=actual_n_models;i++) {
  printf ("Model Number %d\n", i);
  line_has_printing = 0;
  if (beta_0[ i] != 0.0) {
    printf ("%.3f ", beta_0[ i] );
    line_has_printing = 1;
   }
  if (beta_1[ i] > 0.0) {
    printf ("+ ");
   }
  if (beta_1[ i] != 0.0) {
    printf ("%.3fx1 ", beta_1[ i] );
    line_has_printing = 1;
   }
  if (beta_2[ i] > 0.0) {
    printf ("+ ");
   }
  if (beta_2[ i] != 0.0) {
    printf ("%.3fx2 ", beta_2[ i] );
    line_has_printing = 1;
   }
  if (beta_3[ i] > 0.0) {
    printf ("+ ");
   }
  if (beta_3[ i] != 0.0) {
    printf ("%.3fx3 ", beta_3[ i] );
    line_has_printing = 1;
  }
  if (beta_4[ i] > 0.0) {
    printf ("+ ");
   }
  if (beta_4[ i] != 0.0) {
    printf ("%.3fx4 ", beta_4[ i] );
    line_has_printing = 1;
   }
  if (line_has_printing) {
    printf ("\n");
    line_has_printing = 0;
   }
  if (beta_12[ i] > 0.0) {
    printf ("+ ");
   }
  if (beta_12[ i] != 0.0) {
    printf ("%.3fx1x2 ", beta_12[ i] );
    line_has_printing = 1;
   }
  if (beta_13[ i] > 0.0) {
    printf ("+ ");
```

```
 }
if (beta_13[ i]  != 0.0) {
  printf ("%.3fx1x3 ", beta_13[ i] );
  line_has_printing = 1;
 }
if (beta_14[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_14[ i]  != 0.0) {
  printf ("%.3fx1x4 ", beta_14[ i] );
  line_has_printing = 1;
 }
if (beta_23[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_23[ i]  != 0.0) {
  printf ("%.3fx2x3 ", beta_23[ i] );
  line_has_printing = 1;
 }
if (beta_24[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_24[ i]  != 0.0) {
  printf ("%.3fx2x4 ", beta_24[ i] );
  line_has_printing = 1;
 }
if (beta_34[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_34[ i]  != 0.0) {
  printf ("%.3fx3x4 ", beta_34[ i] );
  line_has_printing = 1;
 }
if (line_has_printing) {
  printf ("\n");
  line_has_printing = 0;
 }
if (beta_123[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_123[ i]  != 0.0) {
  printf ("%.3fx1x2x3 ", beta_123[ i] );
  line_has_printing = 1;
 }
if (beta_124[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_124[ i]  != 0.0) {
  printf ("%.3fx1x2x4 ", beta_124[ i] );
  line_has_printing = 1;
 }
if (beta_134[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_134[ i]  != 0.0) {
```

```
  printf ("%.3fx1x3x4 ", beta_134[ i] );
  line_has_printing = 1;
 }
if (beta_234[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_234[ i] != 0.0) {
  printf ("%.3fx2x3x4 ", beta_234[ i] );
  line_has_printing = 1;
 }
if (line_has_printing) {
  printf ("\n");
  line_has_printing = 0;
 }
if (beta_1234[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_1234[ i] != 0.0) {
  printf ("%.3fx1x2x3x4 ", beta_1234[ i] );
  line_has_printing = 1;
 }
if (beta_11[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_11[ i] != 0.0) {
  printf ("%.3fx1**2 ", beta_11[ i] );
  line_has_printing = 1;
 }
if (beta_22[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_22[ i] != 0.0) {
  printf ("%.3fx2**2 ", beta_22[ i] );
  line_has_printing = 1;
 }
if (beta_33[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_33[ i] != 0.0) {
  printf ("%.3fx3**2 ", beta_33[ i] );
  line_has_printing = 1;
 }
if (beta_44[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_44[ i] != 0.0) {
  printf ("%.3fx4**2 ", beta_44[ i] );
  line_has_printing = 1;
 }
if (line_has_printing) {
  printf ("\n");
  line_has_printing = 0;
 }
if (beta_111[ i] > 0.0) {
  printf ("+ ");
```

```
 }
if (beta_111[ i] != 0.0) {
  printf ("%.3fx1**3 ", beta_111[ i] );
  line_has_printing = 1;
 }
if (beta_222[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_222[ i] != 0.0) {
  printf ("%.3fx2**3 ", beta_222[ i] );
  line_has_printing = 1;
 }
if (beta_333[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_333[ i] != 0.0) {
  printf ("%.3fx3**3 ", beta_333[ i] );
  line_has_printing = 1;
 }
if (beta_444[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_444[ i] != 0.0) {
  printf ("%.3fx4**3 ", beta_444[ i] );
  line_has_printing = 1;
 }
if (line_has_printing) {
  printf ("\n");
  line_has_printing = 0;
 }
if (beta_1111[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_1111[ i] != 0.0) {
  printf ("%.3fx1**4 ", beta_1111[ i] );
  line_has_printing = 1;
 }
if (beta_2222[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_2222[ i] != 0.0) {
  printf ("%.3fx2**4 ", beta_2222[ i] );
  line_has_printing = 1;
 }
if (beta_3333[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_3333[ i] != 0.0) {
  printf ("%.3fx3**4 ", beta_3333[ i] );
  line_has_printing = 1;
 }
if (beta_4444[ i] > 0.0) {
  printf ("+ ");
 }
if (beta_4444[ i] != 0.0) {
```

```
      printf ("%.3fx4**4 ", beta_4444[ i] );
      line_has_printing = 1;
     }
    if (line_has_printing) {
      printf ("\n");
     }
    printf ("\n");
   }
 }  /* end PRINT_COEFFS */
```

G.2 Program Listing of C++ Version of EX45

Part of the analysis of the RWWM included investigation of abstractions of the launch point calculation. The following segment of code was written as an alternative approach to the RWWM EX45 subroutine.

```
/* February 22, 1995
C++ Version of RWWM module EX45, Modified to Loop to Demonstrate Effect of
Different Iteration Rates and Allowed Miss Distances .

Note that this module was kept as close as possible to the original Fortran code, no
attempt was made to cleanup or introduce efficiencies
*/
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
FILE *ex45_op                ; /* Pointer to o/p file being used        */
int main()
{
/*
Here we are just defining the parameters, and setting up the arrays.  We put in a
nominal capability for 2 Weasels and 5 sites each.  Each of those sites has at least
5 ARPEOB parameters, but this program only cares about the 5th one, APR range,
and the 2nd one, the site number .
*/
   int wwnum, i, snum, msn,abort  ;
   int smax = 4              ;
   float miss                              ; /* Allowed Miss,  nominally 1000 ft
       */
   float qt10                        ; /* Iteration Rate, nominally 10 per            */
   float wesxyz[ 2][ 3]        ; /*  [ weasel# ][ x, y, alt ]                    */
   float airspd[ 2]                  ; /*  [ weasel# ]                                 */
   int apreob[ 2][ 5][ 5]            ; /*  [weasel#][ 5sites][ -,site#,-,-, range]     */
   float deltat, aprmg, xrng, alt, lang, airsped, ang, deltar, r, pulrate, error ;
/*
Here we had to add a bit of code to initialize parameters that would normally be set
by other modules.  We picked nominal but hopefully realistic values.  Noted that
since AIRSPD in the Fortran is multiplied by 6078 to give AIRSPED in ft/sec, AIRSPD must have
been in miles per second, an unusual choice.
*/
for ( wwnum = 0; wwnum < 2; wwnum++)
    {                                                                  ;/*
      |ww#1|ww2 | */
    airspd[ wwnum]     = 250*(wwnum+1)/3600. ;/* airspd=    |250 |500     | */
    wesxyz[ wwnum][ 1] = 0                          ;/* x coor=    | 0    | 0       | */
    wesxyz[ wwnum][ 2] = 0                                 ;/* y coor=    | 0    | 0       | */
```

```
    wesxyz[ wwnum][ 3] = 5000* (wwnum+1)       ;/* altitud=   | 5k  | 10k    | */
    for (i = 0; i < 5; i++)
      {
      apreob[ wwnum][ i][ 1] = i+1             ;/* site #s are 1, 2, 3, 4,and 5*/
      apreob[ wwnum][ i][ 4] = 2 *( i+1)       ;/* ranges are 2,4,6,8,10 mi  */
      }
    }
  wwnum       =  0                  ; /* For now consider only Weasel #1 */
  msn         =  3                  ; /* - and tell it the target is site #3     */
  pulrate     =  3.5                ; /* Give it a pull-up rate, 3.5 deg/sec     */
  /*
  Here we had to provide code to cause the C++ program to output to a file
  instead of to the screen.  It also causes an abort if the file cannot be accessed.
 */
  if ((ex45_op = fopen("ex45_op.tx", "w")) == NULL)
     {
     printf("Error opening text file for writing\n")                              ;
     exit(0)                                                                      ;
     }
  fprintf(ex45_op,"\tIteration\tAllowed\t\tActual\t\tFinal\tIterations\n")        ;
  fprintf(ex45_op,"\tRate\t\tMiss\t\t\Miss\t\tAngle\tNeeded\n\n")                ;
  fclose(ex45_op)                                                               ;
  printf(          "\tIteration\tAllowed\t\tActual\t\tFinal\tIterations\n")     ;
  printf(          "\tRate\t\tMiss\t\tMiss\t\tAngle\tNeeded\n")                ;
for (qt10 =  0.025;  qt10<1;  qt10 = 2.*qt10)
{
for (miss =  250 ; miss <= 2000; miss = 2* miss)
{
/*_____________________________________________________________________
              The following code mimics the original :
*/
   deltat = 0.0                                          ;
   deltar = 0.0                                                ;
   r = 0.0                                                     ;
   abort = 0                                                   ;
   alt = wesxyz[ wwnum][ 3]                          ;
   lang = 0.0                                                  ;
   airsped = airspd[ wwnum] * 6078                   ;
   xrng = 1.789*alt + 986.*lang + 25766.22  ;
/*
Find the APR-38 site number for this site, get the APR range,  and convert it to
feet.
( Added note:  Minor change since C++ arrays index from 0, Fortran from 1.
Also, since apreob was compared to msn, we had to deduce that the whole array
was integer, not float.)
*/
   for ( i = 0 ; i <= smax ; i++ )
   {
       if  ( apreob[ wwnum][ i][ 1] == msn )
           {
              snum = i                                         ;
              goto six                                         ;
           }
    }
six: aprrng = 6076.0*apreob[ wwnum][ snum][ 4]  ;
/*
Now enter a do loop which checks to see if the X is within  + or - allowed miss
distance (miss) of the target.   If it is the release event is entered in the event list
and we drop out of the loop.  Otherwise increment the weasel's position and
recompute the X's position and do it again.
*/
```

```
   for ( i = 1 ; i <= 95 ; i++ )
     {
     if ((xrng > (aprrng - miss)) &&  (xrng < (aprrng + miss)))
          {
          error = xrng - aprrng                                    ;    /*  We  added  this  for
output*/
          goto eleven                                              ;
          }
     else if ( xrng < aprrng -miss)
          lang = lang + pulrate * qt10                    ;
          else
          lang = lang - pulrate * qt10                    ;
/*
Check to see if launch angle is out of parameters and if so drop out of loop
and abort
*/
     if ((lang < -20.) || (lang > 45.))
          {
       abort = 1                                                   ;
       if (abort == 1)              /* Added this line to stop a nuisance warning */
       goto eleven                                                 ;
          }
/*
If launch angle (lang) has not gone out of parameters, use incremented lang
to calculate an incremented altitude, APR range, range, and X range.  Also
increment time (deltat) to match.
*/

     alt       = alt + (sin(lang/57.3) * airsped * qt10)    ;
     deltar    = cos (lang/57.3) * airsped * qt10           ;
     aprrng    = aprrng - deltar                                        ;
     r         = r + deltar                                 ;
     xrng      = 1.789 *alt  +986.*lang  +25766.22  ;
     deltat    = deltat  + qt10                             ;

     }               /*      This is the end of the 1< i < 95 loop*/
/*
If he aborted, update his current position and return.  Otherwise, compute
the release point and call AR45PK to compute the PK.  First convert the range
travelled back to nautical miles and put the time he aborted into release time for
use by RECOVR to compute the recover points.
*/
eleven:   r = r/6076.
/*
The following is code to output results to file and to screen :
*/                              ;
    if ((ex45_op = fopen("ex45_op.tx", "a")) == NULL)
       {
       printf("Error opening text file for writing\n");
       exit(0);
       }
fprintf(ex45_op,"\t%1.3f\t\t%4.1f\t\t%4.1f\t\t%5.2f\t\t%d\n", qt10, miss,error, lang, i) ;
printf(         "\t%1.3f\t\t%4.1f\t\t%4.1f\t\t%5.2f\t\t%d\n", qt10, miss,error, lang, i) ;
fclose(ex45_op);
}                    /*      This is the end of the "miss" loop   */

}                    /*      This is the end of the "QT10" loop   */


    return 0;
}
```

*U.S. GOVERNMENT PRINTING OFFICE: 1996-509-127-47056

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.